UNIVERSITÀ DEGLI STUDI DI PADOVA

Department of Civil, Environmental and Architectural Engineering

Master's degree in Mathematical Engineering

MASTER THESIS

Order-Flow toxicity and microstructure of crypto order books

Supervisor:
Prof. Grasselli Martino

Candidate: Alessio Cianini 2058085

Co-Supervisor:
Prof. Busca Jerome

ANNO ACCADEMICO 2022-2023

Abstract

The main goal of this thesis is to dive deep into how the order book for Bitcoin actually works. We want to understand how the market behaves under pressure and gain valuable insights into its dynamics.

To achieve this, we'll be using a measure called the VPIN indicator [9], which will help us to figure out how toxic the order flow is.
In this context, toxicity refers to the potential adverse selection and information asymmetry, which can be indicative of trading strategies that capitalize on delayed public information.
This indicator will give us a glimpse into how market participants react and adapt when there is an imbalance between the volume of buy and sell orders.

We'll also explore different models to see which ones can accurately capture the complexities of the Bitcoin market. These models will take into account factors like liquidity, price impact, and market depth. By comparing these models, we hope to improve our understanding of how the Bitcoin market really works.

In addition to analyzing the order book and its statistical properties [4], we'll be studying real trades to get a complete picture of what's happening in the market. We'll use the self-exiting Hawkes process [16] to model the trade data, which allows us to observe patterns and clusters of activity over time. This analysis will give us valuable insights into how trades are connected and influence each other.

The primary objective of this thesis is to predict Bitcoin's price returns.
Initially, we employed an ordinary least squares (OLS[19]) model as a foundational

baseline and then we implemented a Neural Network [27] to see if using a more advanced model we could get better results.

Acknowledgements

I extend my deep gratitude to Prof. M. Grasselli and Prof. Jerome Busca for their guidance and enduring support throughout my thesis. Their valuable insights have greatly contributed to my academic growth.

To my family's unwavering moral support which has been my constant source of strength. I am endlessly thankful for their encouragement and the confidence they have placed in me.

Special thanks to my grandparents, who have been an incredible source of inspiration and encouragement. Their support has been one of my greatest motivators.

To my brother, I offer my heartfelt thanks for being such a solid presence in my life. His kindness and encouragement have been invaluable.

I would also like to express my appreciation to my closest friends — Allegra B., Alberto M., Stefano M., and Tommaso P. — for their companionship and support during my studies. Their presence has been a cherished part of my journey.

And finally, I am grateful to all my other friends who have been part of my life's journey during these years.

Contents

1 Introduction

The primary aim of this Master's thesis is to predict specific market behaviors, particularly drastic changes in prices, and leverage this knowledge for potential benefit.

This prediction of market behaviors will be based on various mathematical models designed to quantify the current state of the market with respect to factors such as order flow toxicity (which is the combination of the risk of adverse selection and information asymmetry that exploit delayed public information), frequency of orders in the market, and market depth.

For addressing order flow toxicity, we employ an indicator named VPIN (The Volume Synchronized Probability of Informed Trading) [9]. VPIN serves the purpose of calculating the imbalance between buy and sell orders. It has been proved to be effective in predicting price crashes in the stock market. This research seeks to examine if similar results can be obtained for cryptocurrencies. The efficacy of the VPIN indicator is illustrated by the 2010 crash for the E-mini S&P, as depicted in Figure 1:

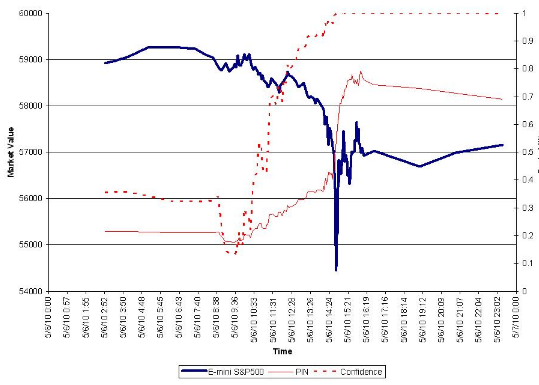


Figure 1: The VPIN indicator during the 2010 Flash Crash for the E-mini S&P

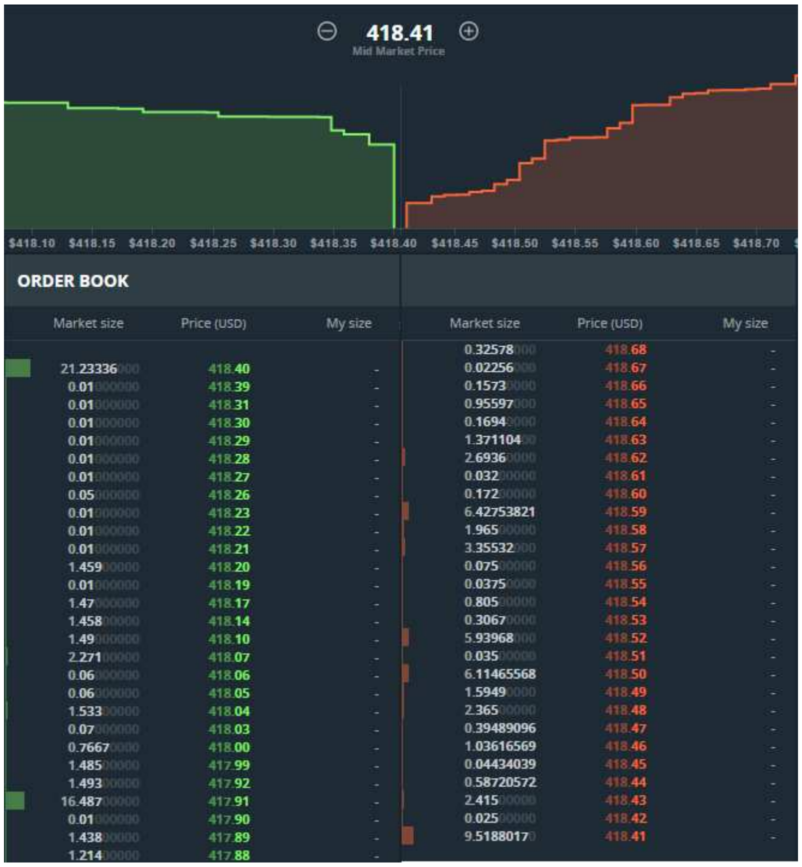
`https://en.wikipedia.org/w/index.php?title=File%3AFlash_Crash.jpg`

Concerning the mathematical model, we aim to validate certain statistical properties of the order book [4] within the cryptocurrency market. Additionally, trades will be modeled using a Hawkes process [11], a popular self-exciting process often employed for understanding the order flow of trades.

The conclusions of this research will be split into two parts. Initially, we will evaluate the accuracy of the VPIN and Hawkes intensity indicators in isolation. Subsequently, we will introduce an additional parameter known as the market depth ratio.

The limit order book comprises a list of buy and sell orders for different prices, with the quantity at each price level forming the volume for that level.

To be more precise in the order book, buy orders are listed on the "bid" side, while sell orders are listed on the "ask" side. Each order includes the price at which the buyer or seller is willing to transact and the quantity of the asset they want to buy or sell. The orders are typically arranged in ascending order on the bid side, with the highest bid price at the top, and in descending order on the ask side, with the lowest ask price at the top. This concept is exemplified in the following figure:



ORDER BOOK

Market size	Price (USD)	My size
21.23336000	418.40	-
0.01000000	418.39	-
0.01000000	418.31	-
0.01000000	418.30	-
0.01000000	418.29	-
0.01000000	418.28	-
0.01000000	418.27	-
0.05000000	418.26	-
0.01000000	418.23	-
0.01000000	418.22	-
0.01000000	418.21	-
1.45900000	418.20	-
0.01000000	418.19	-
1.47000000	418.17	-
1.45800000	418.14	-
1.49000000	418.10	-
2.27100000	418.07	-
0.06000000	418.06	-
0.06000000	418.05	-
1.53300000	418.04	-
0.07000000	418.03	-
0.76670000	418.00	-
1.48500000	417.99	-
1.49300000	417.92	-
16.48700000	417.91	-
0.01000000	417.90	-
1.43800000	417.89	-
1.21400000	417.88	-

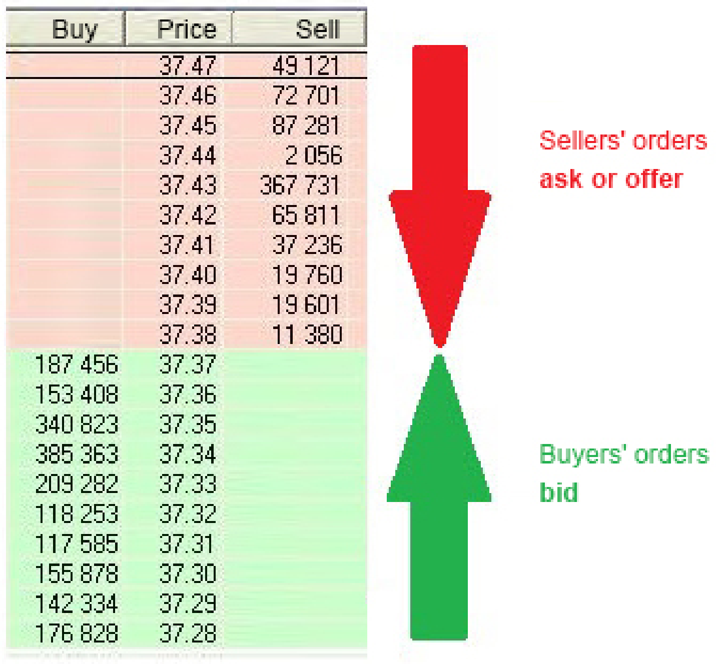
Market size	Price (USD)	My size
0.32578000	418.68	-
0.02256000	418.67	-
0.15730000	418.66	-
0.95597000	418.65	-
0.16940000	418.64	-
1.37110400	418.63	-
2.69360000	418.62	-
0.03200000	418.61	-
0.17200000	418.60	-
6.42753821	418.59	-
1.96500000	418.58	-
3.35532000	418.57	-
0.07500000	418.56	-
0.03750000	418.55	-
0.80500000	418.54	-
0.30670000	418.53	-
5.93968000	418.52	-
0.03500000	418.51	-
6.11465568	418.50	-
1.59490000	418.49	-
2.36500000	418.48	-
0.39489096	418.47	-
1.03616569	418.46	-
0.04434039	418.45	-
0.58720572	418.44	-
2.41500000	418.43	-
0.02500000	418.42	-
9.51880170	418.41	-

Figure 2: Order book ask and bid orders for BTC in 2016

`https://monzo.com/blog/2016/01/08/how-does-the-wholesale-foreign-exchange-market-work`

The green and red areas in the upper part of the figure represent the aggregated volumes for each price leve for BTC in January 2016. Our research will employ an indicator that calculates the ratio of these volumes at each time instance, t_0, under analysis.

If we want to visualize it more linearly this is another image that can help understand how orders are inserted in the market:



Buy	Price	Sell
	37.47	49 121
	37.46	72 701
	37.45	87 281
	37.44	2 056
	37.43	367 731
	37.42	65 811
	37.41	37 236
	37.40	19 760
	37.39	19 601
	37.38	11 380
187 456	37.37	
153 408	37.36	
340 823	37.35	
385 363	37.34	
209 282	37.33	
118 253	37.32	
117 585	37.31	
155 878	37.30	
142 334	37.29	
176 828	37.28	

Sellers' orders
ask or offer
Buyers' orders
bid

Figure 3: Order book vertical

https://www.ifcmarkets.com/en-CA/about-forex/order-book

After analyzing the market variables, we'll employ the Ordinary Least Squares (OLS[19]) model.

As baseline we aim to use OLS to fit our data and develop a model that can forecast price returns.

However, our main interest is in using a Neural Network[27] for predictions.

2 Reconstruction of Order Book and Trades

In my research, I obtain the necessary data by utilizing the Binance API[7], which allows me to download historical tick-by-tick order book events and trade records. This data plays a critical role in conducting microstructural analysis, calculating the VPIN indicator [9], exploring various models, and modeling trades using the Hawkes process [11].

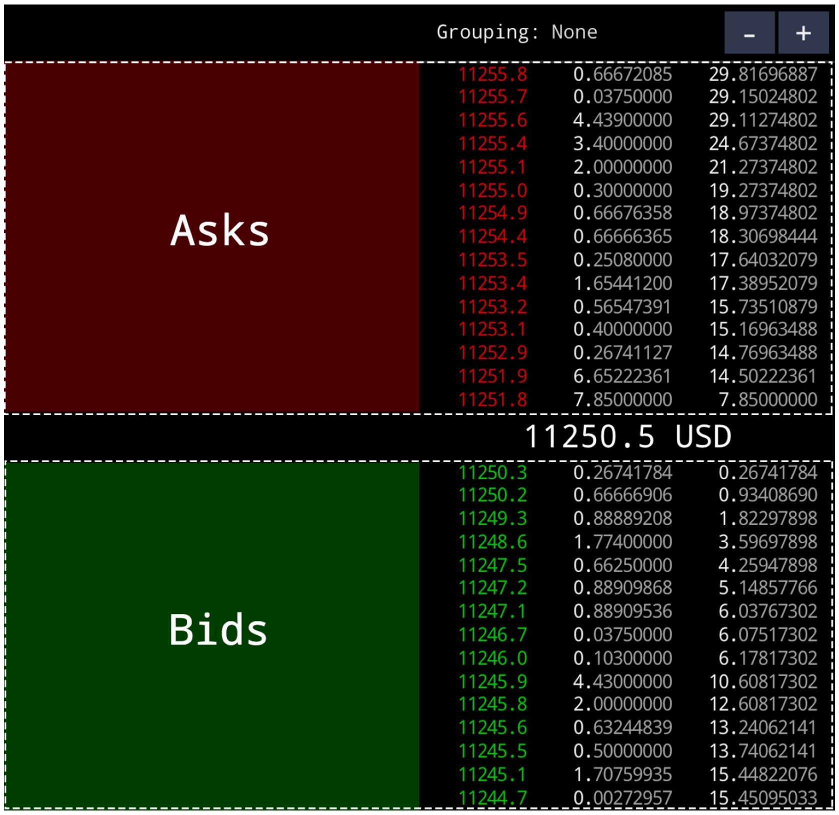
By accessing this data, I gain valuable insights into the dynamics of the market. It should enable to make informed trading decisions and understand how market participants react and adapt to different situations.
The historical order book events and trade records obtained through the Binance API provide a comprehensive view of market activity, which is crucial for conducting detailed analyses and drawing meaningful conclusions.

Having access to this data is instrumental in understanding the microstructure of the Bitcoin market and contributes to a more comprehensive understanding of its underlying dynamics. It allows for robust analysis and helps uncover patterns, trends, and correlations that are essential for making informed decisions in trading and gaining insights into the market's behavior.

2.1 Order book and Trades

2.1.1 Order Book

The order book[15] is a fundamental component of financial markets, including the Bitcoin market. It serves as a centralized record of all buy and sell orders for a specific asset or security, such as Bitcoin, and is organized based on price and time.



11255.8	0.66672085	29.81696887
11255.7	0.03750000	29.15024802
11255.6	4.43900000	29.11274802
11255.4	3.40000000	24.67374802
11255.1	2.00000000	21.27374802
11255.0	0.30000000	19.27374802
11254.9	0.66676358	18.97374802
11254.4	0.66666365	18.30698444
11253.5	0.25080000	17.64032079
11253.4	1.65441200	17.38952079
11253.2	0.56547391	15.73510879
11253.1	0.40000000	15.16963488
11252.9	0.26741127	14.76963488
11251.9	6.65222361	14.50222361
11251.8	7.85000000	7.85000000

11250.5 USD

11250.3	0.26741784	0.26741784
11250.2	0.66666906	0.93408690
11249.3	0.88889208	1.82297898
11248.6	1.77400000	3.59697898
11247.5	0.66250000	4.25947898
11247.2	0.88909868	5.14857766
11247.1	0.88909536	6.03767302
11246.7	0.03750000	6.07517302
11246.0	0.10300000	6.17817302
11245.9	4.43000000	10.60817302
11245.8	2.00000000	12.60817302
11245.6	0.63244839	13.24062141
11245.5	0.50000000	13.74062141
11245.1	1.70759935	15.44822076
11244.7	0.00272957	15.45095033

Figure 4: Vertical representation of the Order Book for list of offers in sell and buy in descending price order

`https://guides.cryptowat.ch/trader-education/order-books-and-market-depth-charts-explained`

The order book provides traders with valuable insights into the supply and demand dynamics of the market in real-time. By examining the order book, traders can see the current best bid and ask prices, which represent the highest price a buyer is willing to pay and the lowest price a seller is willing to accept, respectively. These prices are often referred to as the "top of the book"[15].

Overall, the order book provides traders with real-time visibility into the supply and demand for a specific asset, enabling them to make informed trading decisions based on current market conditions. Understanding the order book is essential for navigating the complexities of financial markets and capitalizing on trading opportunities.

2.1.2 Trades

[29] When the bids and asks order get filled it represents the actual buying and selling of assets at specific prices. When a trade occurs, it reflects the convergence of a buyer and a seller agreeing on a transaction.

Each trade is recorded in the order book, which serves as a central record of all buy and sell orders. When a trade takes place, it results in a change to the order book. Specifically, the buyer's order is filled, meaning they obtain the desired quantity of the asset at the agreed-upon price, and the corresponding sell order from the seller is removed from the order book.

The occurrence of trades provides valuable insights into the real-time supply and demand dynamics of the market. By analyzing the trades, traders can observe the actual prices at which transactions are happening and the corresponding volumes being traded. This information helps them assess market liquidity and understand the current market sentiment.

Trades are essential for market participants to execute their strategies and manage their positions. Traders may enter trades to take advantage of price movements, hedge against risks, or seek profitable opportunities. The ability to monitor and analyze trade data allows traders to make informed decisions based on actual market activity.

In addition to their immediate impact on the order book, trades can also have ripple effects on market dynamics. For example, large trades can result in price movements, as they indicate significant buying or selling pressure. These price movements may, in turn, influence the behavior of other market participants, creating a chain reaction of additional trades and potential market trends.

Understanding the patterns and characteristics of trades is crucial for traders and researchers alike. By studying trade data, one can identify trends, detect market anomalies, and develop trading strategies based on historical trade patterns. Moreover, trade data is often utilized in quantitative analysis and modeling, such as the application of the Hawkes process[11] mentioned earlier, to capture interdependencies and clustering effects within the market.

In summary, trades represent the actual buying and selling of assets at specific prices. They are recorded in the order book and reflect the real-time supply and demand dynamics of the market. Analyzing trade data provides insights into market activity, helps traders make informed decisions, and contributes to a deeper understanding of market behavior and dynamics.

2.1.3 Binance Data

As mentioned earlier, the Binance API[7] allows us to download historical tick-by-tick data, which is essential for our analysis of the order book and trades. This data provides us with the ability to reconstruct the order book at any specific moment and examine the trades that occurred throughout the day.

The data we obtain from the Binance API is typically presented in CSV (Comma-Separated Values) format, which is easy to work with and analyze. For the order book data, each event in the data records a specific quantity at a particular price level. This means that we can see the individual orders placed at different price levels, rather than just the changes in the order book from the previous state. This level of detail helps us to understand the supply and demand dynamics at various price levels.
We can see in order:

- **Symbol -** For example: 'BTCUSDT', 'BTCUSD_200925'.
- **Time -** Transaction time in timestamp.
- **First update id**
- **Last update id**
- **Side**

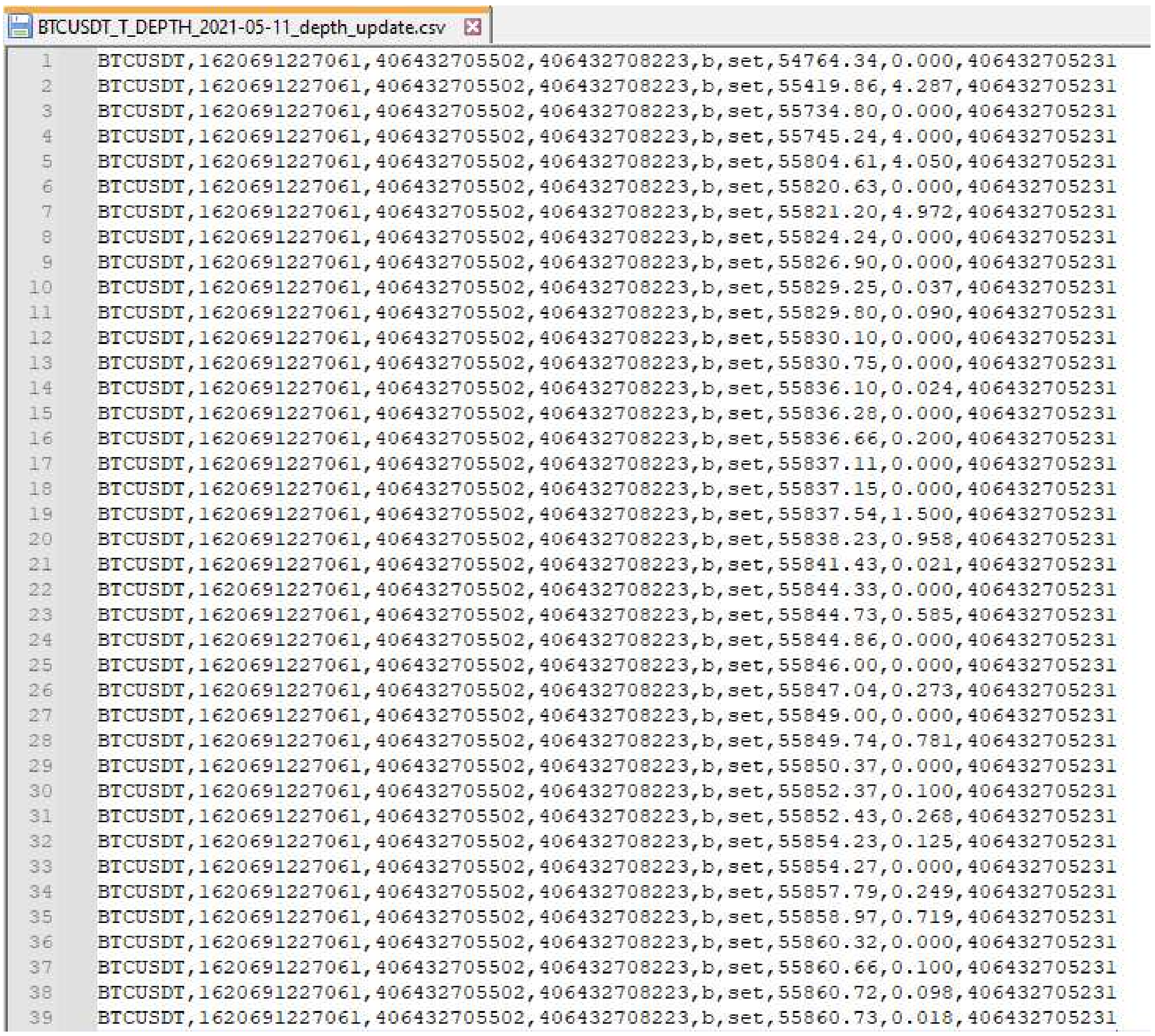
  - a = ask (Sell order)
  - b = bid (Buy order)

- **Update type**
  - snap = for order book snapshot only
  - set = set price level to the current qty (not delta)
  - delta = qty change of the price level (delta)

- **Price**

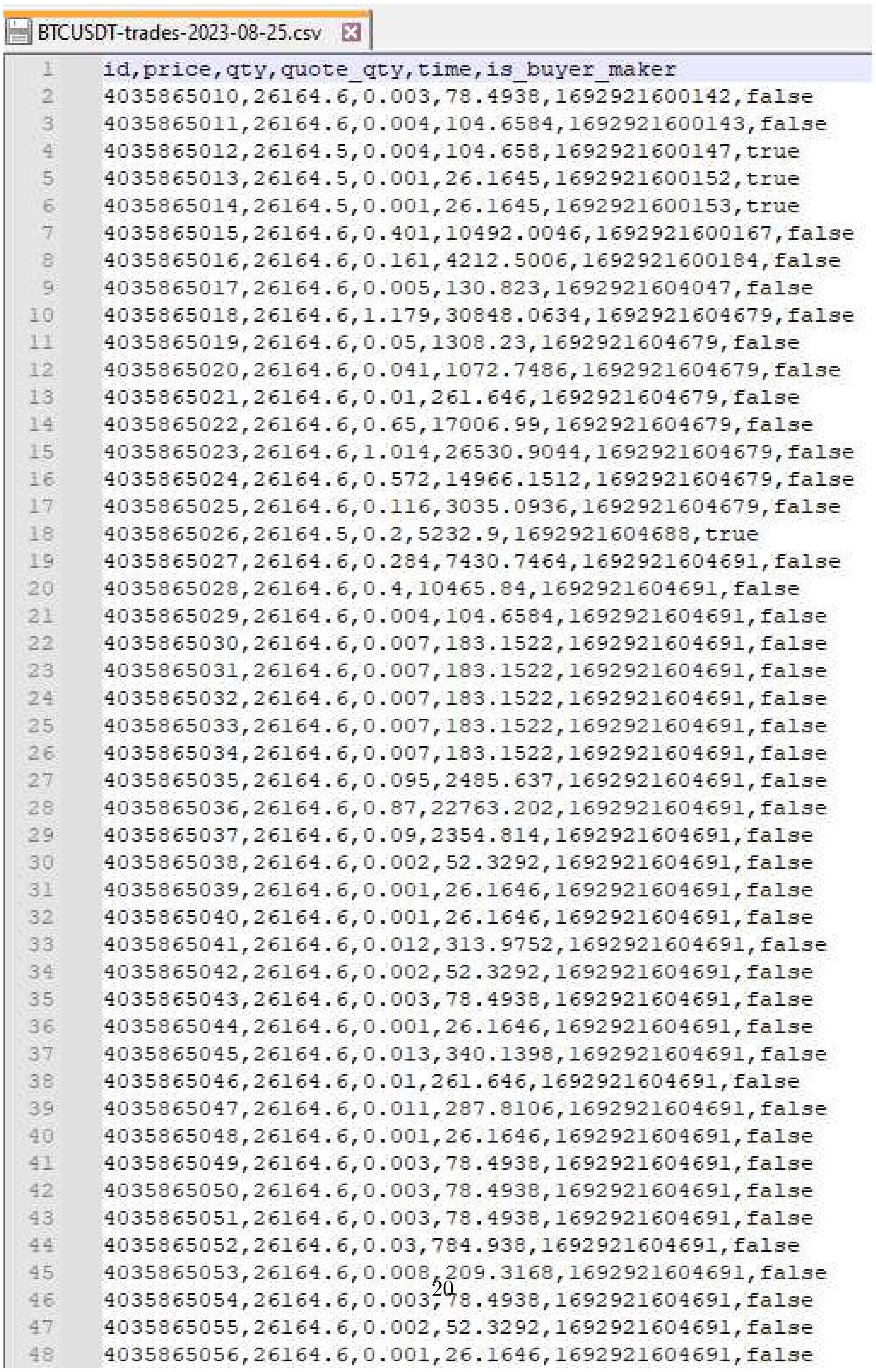
- **Qty**

- **Pu -** Last update id of the previous row, which helps examine completeness. Only applicable to COIN-M Futures depth update.


BTCUSDT_T_DEPTH_2021-05-11_depth_update.csv

```
BTCUSDT,162069122706l,406432705502,406432708223,b,set,54764.34,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55419.86,4.287,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55734.80,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55745.24,4.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55804.61,4.050,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55820.63,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55821.20,4.972,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55824.24,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55826.90,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55829.25,0.037,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55829.80,0.090,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55830.10,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55830.75,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55836.10,0.024,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55836.28,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55836.66,0.200,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55837.11,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55837.15,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55837.54,1.500,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55838.23,0.958,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55841.43,0.021,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55844.33,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55844.73,0.585,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55844.86,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55846.00,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55847.04,0.273,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55849.00,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55849.74,0.781,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55850.37,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55852.37,0.100,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55852.43,0.268,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55854.23,0.125,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55854.27,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55857.79,0.249,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55858.97,0.719,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55860.32,0.000,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55860.66,0.100,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55860.72,0.098,406432705231
BTCUSDT,162069122706l,406432705502,406432708223,b,set,55860.73,0.018,406432705231
```

On the other hand when it comes to trades, the data shows us the total quantity traded at each price level. This information gives us insights into the trading activity that took place at different price points, showing the levels of buying and selling interest in the market.



```
id,price,qty,quote_qty,time,is_buyer_maker
4035865010,26164.6,0.003,78.4938,1692921600142,false
4035865011,26164.6,0.004,104.6584,1692921600143,false
4035865012,26164.5,0.004,104.658,1692921600147,true
4035865013,26164.5,0.001,26.1645,1692921600152,true
4035865014,26164.5,0.001,26.1645,1692921600153,true
4035865015,26164.6,0.401,10492.0046,1692921600167,false
4035865016,26164.6,0.161,4212.5006,1692921600184,false
4035865017,26164.6,0.005,130.823,1692921604047,false
4035865018,26164.6,1.179,30848.0634,1692921604679,false
4035865019,26164.6,0.05,1308.23,1692921604679,false
4035865020,26164.6,0.041,1072.7486,1692921604679,false
4035865021,26164.6,0.01,261.646,1692921604679,false
4035865022,26164.6,0.65,17006.99,1692921604679,false
4035865023,26164.6,1.014,26530.9044,1692921604679,false
4035865024,26164.6,0.572,14966.1512,1692921604679,false
4035865025,26164.6,0.116,3035.0936,1692921604679,false
4035865026,26164.5,0.2,5232.9,1692921604688,true
4035865027,26164.6,0.284,7430.7464,1692921604691,false
4035865028,26164.6,0.4,10465.84,1692921604691,false
4035865029,26164.6,0.004,104.6584,1692921604691,false
4035865030,26164.6,0.007,183.1522,1692921604691,false
4035865031,26164.6,0.007,183.1522,1692921604691,false
4035865032,26164.6,0.007,183.1522,1692921604691,false
4035865033,26164.6,0.007,183.1522,1692921604691,false
4035865034,26164.6,0.007,183.1522,1692921604691,false
4035865035,26164.6,0.095,2485.637,1692921604691,false
4035865036,26164.6,0.87,22763.202,1692921604691,false
4035865037,26164.6,0.09,2354.814,1692921604691,false
4035865038,26164.6,0.002,52.3292,1692921604691,false
4035865039,26164.6,0.001,26.1646,1692921604691,false
4035865040,26164.6,0.001,26.1646,1692921604691,false
4035865041,26164.6,0.012,313.9752,1692921604691,false
4035865042,26164.6,0.002,52.3292,1692921604691,false
4035865043,26164.6,0.003,78.4938,1692921604691,false
4035865044,26164.6,0.001,26.1646,1692921604691,false
4035865045,26164.6,0.013,340.1398,1692921604691,false
4035865046,26164.6,0.01,261.646,1692921604691,false
4035865047,26164.6,0.011,287.8106,1692921604691,false
4035865048,26164.6,0.001,26.1646,1692921604691,false
4035865049,26164.6,0.003,78.4938,1692921604691,false
4035865050,26164.6,0.003,78.4938,1692921604691,false
4035865051,26164.6,0.003,78.4938,1692921604691,false
4035865052,26164.6,0.03,784.938,1692921604691,false
4035865053,26164.6,0.008,209.3168,1692921604691,false
4035865054,26164.6,0.003,78.4938,1692921604691,false
4035865055,26164.6,0.002,52.3292,1692921604691,false
4035865056,26164.6,0.001,26.1646,1692921604691,false
```

We can see in order:

- **Id** - Trade ID
- **Qty** - Quantity
- **Base Qty** - Base Quantity
- **Price** - Price
- **Time** - Time in Unix time format
- **Is Buyer Maker** - Was the buyer the maker

By utilizing the historical tick-by-tick data obtained through the Binance API, we can reconstruct the order book at different points in time and analyze the trades that occurred. This detailed data allows us to identify patterns, trends, and potential trading opportunities.

The availability of this data in a format that captures both the order book events and trade quantities enables us to conduct thorough analysis and modeling of market behavior. We can apply various quantitative techniques and models to gain deeper insights into the microstructure of the market and develop trading strategies based on historical patterns and trends.

In summary, the Binance API provides us with historical tick-by-tick data in CSV format, allowing us to reconstruct the order book and analyze trades. This data gives us specific quantities at each price level for the order book events, and the total traded quantities for each price level in the case of trades. By leveraging this data, we can gain a comprehensive understanding of market dynamics and make informed decisions based on historical market activity.

2.2 Implemention for the riconstruction of the order book with C++

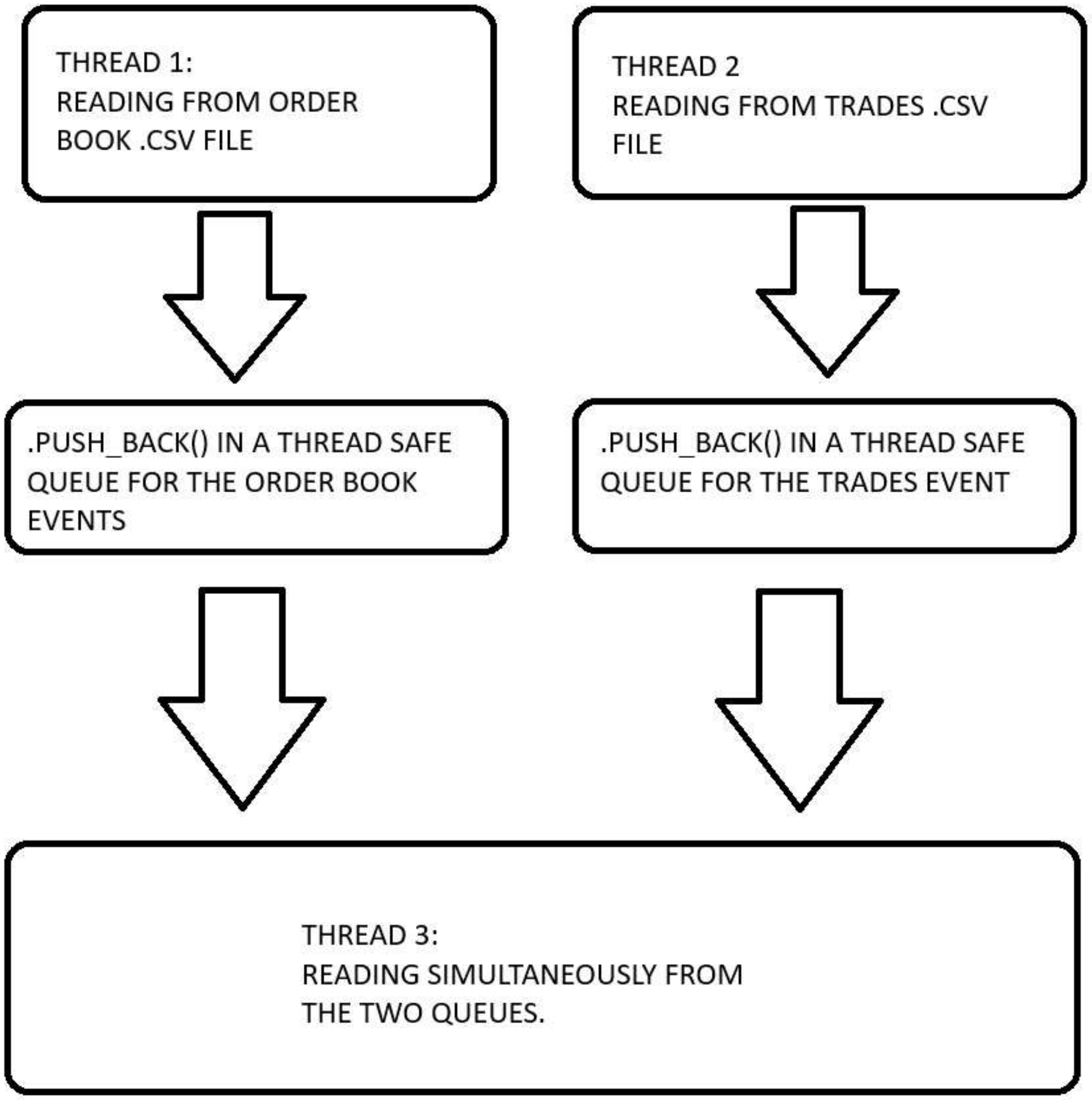
2.2.1 C++ choice and multithreading

Due to the substantial size of the data file for analysis, we opted to use C++ as the programming language to handle the analysis efficiently. To enhance performance, a multithreading approach was adopted, utilizing three distinct threads to streamline the process.

The first thread was designed to read and store the order book data in queue1, a data structure that allows for efficient data storage and retrieval. This thread specifically focused on handling the order book information, ensuring its proper organization and availability for further analysis.

Simultaneously, the second thread was responsible for reading and storing the trade data in queue2. This thread dedicated its efforts to managing the trade data, maintaining its integrity and accessibility throughout the analysis phase.

The third thread played a crucial role in the analysis process. Its primary responsibility was to examine the data from the top of each queue and determine which one had the least timestamp. By comparing the timestamps, the thread could select the data with the earlier occurrence, indicating the chronological order of events. Once the analysis was completed, the corresponding data was removed from the respective queue to optimize memory usage.


THREAD 1:
READING FROM ORDER BOOK .CSV FILE
THREAD 2
READING FROM TRADES .CSV FILE
.PUSH_BACK() IN A THREAD SAFE QUEUE FOR THE ORDER BOOK EVENTS
.PUSH_BACK() IN A THREAD SAFE QUEUE FOR THE TRADES EVENT
THREAD 3:
READING SIMULTANEOUSLY FROM THE TWO QUEUES.

By implementing this multithreading approach, we aimed to improve the efficiency of the analysis process, allowing for simultaneous handling of order book and trade data. The division of tasks among the threads helped streamline the operations, enabling faster data processing and reducing potential bottlenecks.

Overall, the utilization of C++ and multithreading techniques facilitated the effective handling and analysis of the large data file. This approach ensured optimal utilization of system resources, enabling efficient organization, comparison, and removal of data from the queues.

2.2.2 Thread Safe Queue

During the implementation of the multithreading application, we encountered challenges related to shared variables, particularly in the case where both threads accessed the same queue simultaneously. This concurrent access could potentially lead to data inconsistencies, race conditions, or other issues.

To address this challenge and ensure data integrity, we created a custom thread-safe queue class. This class implements synchronization mechanisms to control access to the queue and ensure that only one thread can access it at a time. By enforcing this synchronization, we prevent concurrent modifications and maintain consistency in the queue's state. The thread-safe queue class employs techniques such as mutexes, locks, or other synchronization primitives to manage access to the shared resource. When a thread wants to access the queue, it first acquires a lock or mutex, ensuring exclusive access. Once the thread completes its operation, it releases the lock, allowing other threads to access the queue.

By utilizing this custom thread-safe queue class, we successfully mitigate the risks associated with concurrent access to the shared queue. This approach guarantees that data modifications are performed in a controlled manner, eliminating data inconsistencies and ensuring the reliability of the multithreading application.

Handling shared variables and ensuring thread safety is crucial in multithreaded applications, especially when multiple threads are accessing the same resources. The creation of a custom thread-safe queue class exemplifies our commitment to maintaining data integrity and resolving challenges related to concurrent access in the multithreading implementation.

ThreadSafeQueue.cpp

```cpp
#include "ThreadSafeQueue.h"
ThreadSafeQueue::ThreadSafeQueue() {}
void ThreadSafeQueue::push(std::string item) {
    std::lock_guard<std::mutex> lock(m_mutex);
    m_queue->push(item);
    }
void ThreadSafeQueue::pop() {
    std::lock_guard<std::mutex> lock(m_mutex);
    m_queue->pop();
}
std::string ThreadSafeQueue::front() {
    std::lock_guard<std::mutex> lock(m_mutex);
    std::string item = m_queue->front();
    return item;
}
bool ThreadSafeQueue::is_empty() {
    std::lock_guard<std::mutex> lock(m_mutex);
    return m_queue->empty();
}
size_t ThreadSafeQueue::size() {
    std::lock_guard<std::mutex> lock(m_mutex);
    return m_queue->size();
}
```

ThreadSafeQueue.h

```cpp
#pragma once
#ifndef QUEUETHREAD_H
#define QUEUETHREAD_H
#include <queue>
#include <mutex>
#include <string>

class ThreadSafeQueue {
private:
    std::queue<std::string>* m_queue = new
       std::queue<std::string>;
    std::mutex m_mutex;

public:
    ThreadSafeQueue();
    void push(std::string item);
    void pop();
    std::string front();
    bool is_empty();
    size_t size();

};

#endif
```

2.3 Bouchaud-Mezard-Potters analysis

The first analysis that has been done is a check to see if the shape of the average order book behave like stated in the paper "Statistical properties of stock order books: empirical results and models of Bouchaud-Mezard-Potters"[4].

In the paper is showed the average volume of the queue in the order book as a funcion of the distance Δ from the current bid or ask in a log-linear scale.

In order to achieve the same behaviour for each timestamp an average w.r.t. to the previous average order book shape has been calculated.

The following images will show the average size of the order book boh for bid and ask.

In [Figure 5] we have zoomed in an interval which is close to the mid price:

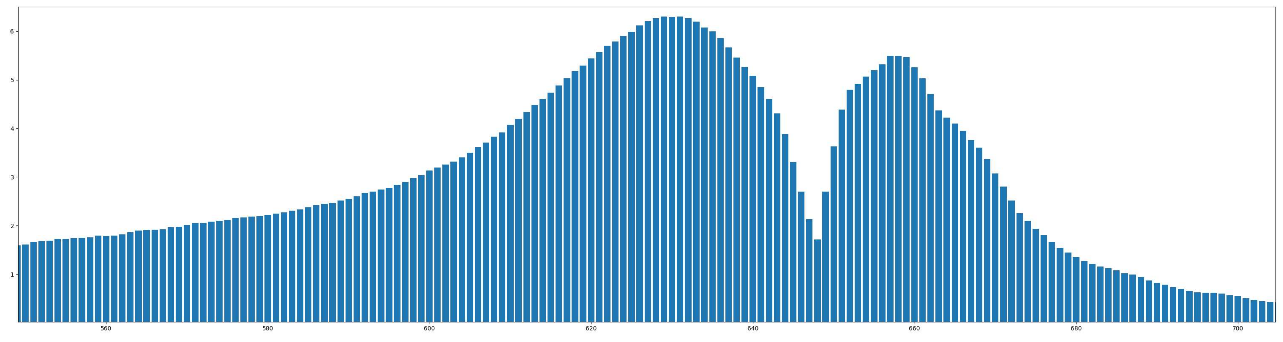


Figure 5: Zoom near the mid price to see the average volume for each price level.

In [Figure 6] we took all the level of the orderbook registered while constructing it:

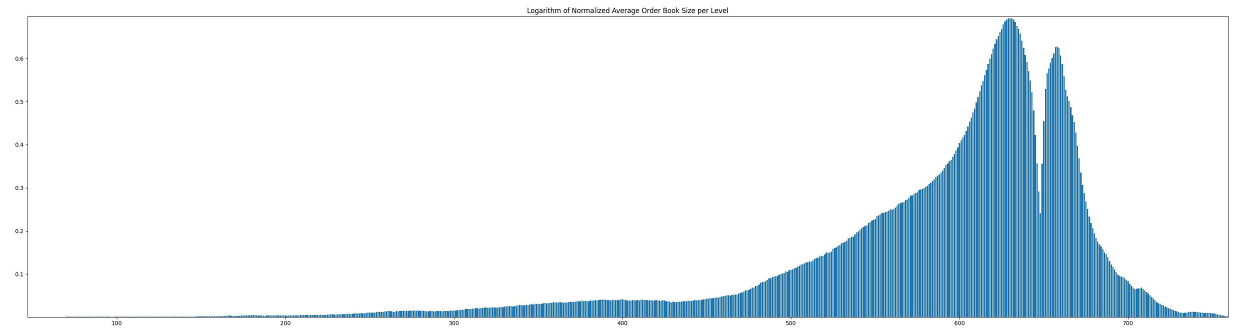


Figure 6: Average volume with all the price levels displayed without zoom.

The result obtained show that the properties stated in the paper even for the cryptocurrency market are satisfied:

From the paper we have that the function that gives the shape of the averaged order book is given by:

$$\rho_{st}(\Delta) = e^{-\Delta} \int_0^{\Delta} duu^{-1-\mu} sinh(u) + sinh(\Delta) \int_0^{\infty} duu^{-1-\mu} e^{-u}$$

And the properties are:

- for the price distance Δ from the mid price if $\Delta \to 0$ we have that the average volume in the order book tends to 0

- whilst the average order book volume reflects the incoming flow of orders: $\rho_{st}(\Delta) \propto \Delta^{-1-\mu}$.

In summary for [4], the form of the average order book is a manifestation of the interplay between a power-law influx of limit orders with a limited duration and the price dynamics that clear orders in proximity to the current market price.

3 Hawkes Process

Before introducing the Hawkes Process, it's necessary to first discuss its foundational concepts, namely the Counting Process, the Poisson Process, and the self-exciting process.

3.1 Counting Process

Definition: A point process is a sequence of non-negative random variables $\{T_i, i \in \mathbb{N}\}$ such that for every $i \in \mathbb{N}$, $T_i < T_{i+1}$.

Definition [17]: A stochastic process ([25], pg.53) $\{N(t), t > 0\}$ is referred to as a counting process if $N(t)$ signifies the total number of events that occurred by time t. The stochastic process is defined as:

$$N(t) = \sum_{i \in \mathbb{N}} I_{\{T_i \leq t\}}$$

where T_i is a point process.

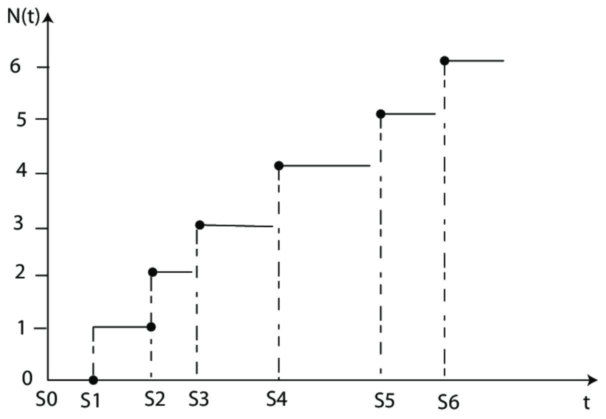


Figure 7: Sample path realization of counting process

`https://www.researchgate.net/figure/Sample-path-realization-of-counting-process_fig4_265081596`

3.1.1 Poisson Process

Definition: The counting process $\{N(t), t \geq 0\}$ is termed a Poisson Process with rate $\lambda > 0$ if the following properties are satisfied:

1. $N(0) = 0$
2. $\{N(t), t \geq 0\}$ has independent increments
3. $\mathbb{P}(N(t+h) - N(t) \geq 2) = o(h)$
4. $\mathbb{P}(N(t+h) - N(t) = 1) = \lambda h + o(h)$

Here, $o(h)$ is a part of the Landau Notation [30] for Big O and Little o.

Theorem ([22], pg.312) If $\{N(t), t > 0\}$ is a Poisson Process with rate $\lambda > 0$, then for all $s > 0$ and $t > 0$, $N(s+t) - N(s)$ is a Poisson random variable with mean λt.

3.1.2 Non-homogeneous Poisson Process

Definition: The counting process $\{N(t), t \geq 0\}$ is known as a non-homogeneous Poisson Process with an intensity function $\lambda(t), t > 0$ if it satisfies:

1. $N(0) = 0$
2. $\{N(t), t \geq 0\}$ has independent increments
3. $\mathbb{P}(N(t+h) - N(t) \geq 2) = o(h)$
4. $\mathbb{P}(N(t+h) - N(t) = 1) = \lambda(t)h + o(h)$

In this case, the intensity function for a non-homogeneous Poisson Process is deterministic.

3.1.3 Self-Exciting process

3.1.3.1 Self-exciting definition

From [11], considering a point process that respects properties (3) ad (4) for a Poisson Process we have that the process is self-exciting in the sense that :

$$\Lambda(t) = \nu + \int_{-\infty}^{t} g(t-u)dN(u),$$

As described in [11], this can be interpreted as a self-exciting shot process, where the current intensity of events is influenced by past events. We assume that $g(\nu) \geq 0$ and $g(\nu = 0)$ $\nu < 0$.

If we assume sationarity then from the definition we express Λ:

$$\lambda = \mathbb{E}[\lambda(t)] = \nu + \int_{-\infty}^{t} g(t-u)du,$$

or

$$\lambda = \frac{\nu}{\left[1 - \int_0^\infty g(v)dv\right]}.$$

3.2 Simulation of Hawkes Process with C++ and Python

[18] proposed an algorithm for the simulation of Hawkes processes.

Let us denote by UNIF$[0, 1]$ the uniform distribution on the interval $[0, 1]$ and $[0, T]$ the time interval on which the process is to be simulated.

We assume here that $P = 1$.

1. **Initialization:** Set $\lambda^* \leftarrow \lambda_0(0)$, $n \leftarrow 1$.
2. **First event:** Generate $U \sim$ UNIF$[0, 1]$ and set $s \leftarrow -\frac{1}{\lambda^*} \ln U$.
    - If $s \leq T$, then set $T_1 \leftarrow s$.
    - Else, go to the last step.
3. **General routine:** Set $n \leftarrow n + 1$.
    (a) **Update maximum intensity:** Set $\lambda^* \leftarrow \lambda(T_{n-1}) + Y$. λ^* exhibits a jump of size Y as an event has just occurred.
    (b) **New event:** Generate $U \sim$ UNIF$[0, 1]$ and set $s \leftarrow s - \frac{1}{\lambda^*} \ln U$.
        - If $s \geq T$, then go to the last step.
    (c) **Rejection test:** Generate $D \sim$ UNIF$[0, 1]$. If $D \leq \frac{\lambda(s)}{\lambda^*}$, then set $T_n \leftarrow s$, and go through the general routine again. Else, update $\lambda^* \leftarrow \lambda(s)$ and try a new date at step (b) of the general routine.
4. **Output:** Retrieve the simulated process $\{T_n\}$ on $[0, T]$.

Intensity Function

```cpp
double intensity(double t, const std::vector<double>&
    arrivals)
{
 double lambdaValue = mu;
 for (double arrival : arrivals) {
   lambdaValue += alpha * std::exp(-beta * (t - arrival));
 }
 return lambdaValue;
}
```

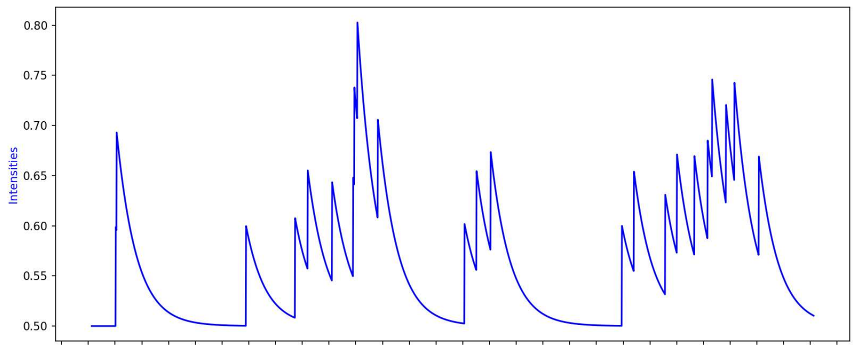
Simulation with [18] Method

```
std::vector<double> simulateHawkes\cite{6}(double T) {
  std::random_device rd;
  std::default_random_engine generator(rd());
  std::uniform_real_distribution<double>
      uniformDistribution(0.0, 1.0);
  std::vector<double> events;
  double s = 0.0;
  std::vector<double> arrivals;
  int n = 0;
  while (s < T)
  {
    n++;
    double lambdaStar = intensity(s, arrivals);
    double U = uniformDistribution(generator);
    s -= (1 / lambdaStar) * std::log(U);
    if (s >= T) {
      break;
    }
    double D = uniformDistribution(generator);
    double currentIntensity = intensity(s, arrivals);
    if (D <= currentIntensity / lambdaStar)
    {
      events.push_back(s);
      arrivals.push_back(s);
    }
  }
  return events;
}
```

With the method described in [18] we can simulate the Hawkes Process using these parameters for the intensity :

$$\alpha = 0.1 \quad \beta = 1.5 \quad \mu = 0.5$$

and in an interval from T = 0 to T = 25 will lead us to:



This Figure show the exponential decay observed between successive points.

3.3 Expectation Maximization algorithm for Hawkes Process

The Expectation Maximization (EM) algorithm is a cyclic process utilized for calculating maximum likelihood approximations when perceived observations can be regarded as incomplete data [5]. The algorithm works in two primary steps: the Expectation step (E-step), and the Maximization step (M-step). During the E-step, latent variables are assessed based on observed data and current parameter approximations. Then, during the M-step, the likelihood function is optimized using the latent variables estimated in the preceding E-step, this stage derives new approximations for the upcoming iteration. With each cycle, the likelihood escalates, ensuring convergence [3]. [28] proposed applying the EM algorithm to Hawkes processes, treating the unobservable branching structure under the Poisson cluster representation as latent variables for parameter estimation. However, the computational load of the EM algorithm can be significant, so various approximations are often implemented to make estimation less resource-intensive.

3.3.1 Mathematical Approach

From [16] we consider a Hawkes Process with intesity:

$$\lambda(t) = \mu(t) + \sum_{t>t_i} g(t - t_i)$$

with exponential triggering kernel $g(t) = \alpha\omega e^{-\omega t}$ on the time interval $[0, T]$ and the EM algorithm proposed in [28] for the estimation of the parameters $\Theta = (\mu, \alpha, \omega)$ from a point sample $\{t_i\}_{i=1}^{n}$ generated according to the intensity. The log-likelihood function is given by

$$l(\Theta) = \sum_{i=1}^{n} \log(\lambda(t_i)) - \int_0^T \lambda(t)dt \tag{1}$$

where the integral approximation, $\mu T + \alpha \sum_{i=1}^{n}(1 - e^{-\omega(T-t_i)}) \approx \mu T + \alpha n$, can be used when $\omega^{-1} \ll T$. Let

$$A(i) = \sum_{j=1}^{i-1} e^{-\omega(t_i - t_j)}$$
$$B(i) = \sum_{j=1}^{i-1} (t_i - t_j) e^{-\omega(t_i - t_j)}$$

Then the derivatives of the log-likelihood with respect to the parameters are given by

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^{n} \frac{1}{\lambda(t_i)} - T$$

$$\frac{\partial l}{\partial \alpha} = \sum_{i=1}^{n} \frac{\omega A(i)}{\lambda(t_i)} - n$$

$$\frac{\partial l}{\partial \omega} = \sum_{i=1}^{n} \frac{\alpha}{\lambda(t_i)} (A(i) - \omega B(i))$$

The EM algorithm for the estimation of the parameters is as follows. Let p_{ij} be the probability that event j triggers event i. Starting with a guess Θ^0 for the parameters, iterate the following until convergence is reached:

Expectation step:

$$p_{ij}^k = \frac{\alpha^k \omega^k e^{-\omega^k (t_i - t_j)}}{\mu^k + \sum_{j=1}^{i-1} \alpha^k \omega^k e^{-\omega^k (t_i - t_j)}}$$

$$p_{ii}^k = \frac{\mu^k}{\mu^k + \sum_{j=1}^{i-1} \alpha^k \omega^k e^{-\omega^k (t_i - t_j)}}$$

Maximization step:

$$\mu^{k+1} = \frac{\sum_{i=1}^{n} p_{ii}^k}{T}$$

$$\alpha^{k+1} = \frac{\sum_{i>j} p_{ij}^k}{n}$$

$$\omega^{k+1} = \frac{\sum_{i>j} p_{ij}^k}{\sum_{i>j} (t_i - t_j) p_{ij}^k}$$

It is easy to show that each EM iteration satisfies the following:

$$\mu^{k+1} - \mu^k = \frac{\mu^k}{T} \frac{\partial l}{\partial \mu}$$

$$\alpha^{k+1} - \alpha^k = \frac{\alpha^k}{n} \frac{\partial l}{\partial \alpha}$$

$$\omega^{k+1} - \omega^k = \frac{\omega^k}{\sum_{i>j} (t_i - t_j) p_{ij}^k} \frac{\partial l}{\partial \omega}$$

The EM algorithm is essentially a projected gradient ascent method when you look at the point where partial derivatives are evaluated at Θ^k. This observation was also made by "Xu & Jordan (1996)" [33] in their study of EM algorithm for Gaussian mixtures. They found that this projection technique enhances the rate of convergence, especially when there's minimal overlap in the mixtures.

For the sake of simplicity, we'll express equations (11)-(13) as:

$$\Theta^{k+1} - \Theta^k = P(\Theta^k)\nabla l|_{\Theta^k}$$

In this equation, $P(\Theta^k)$ is a diagonal matrix with all entries positive. If Θ^* represents the MLE estimate of (2), we can use Taylor's expansion to express the gradient of the log-likelihood about Θ^*:

$$\|\Theta^{k+1} - \Theta^*\| \leq \|I + P(\Theta^k)H(\Theta^*)\|\|\Theta^k - \Theta^*\|$$

Here, H stands for the Hessian of the log-likelihood. This tells us that the rate at which the EM algorithm converges is contingent upon the condition of the matrix PH.

When dealing with boundary correction, we need to consider when the integral in (2) is computed precisely. The partial derivatives of the log-likelihood with respect to α and ω would then be:

$$\frac{\partial l}{\partial \alpha} = \sum_{i=1}^{n} \frac{\omega A(i)}{\lambda(t_i)} - n + \sum_{i=1}^{n} e^{-\omega(T-t_i)}$$

$$\frac{\partial l}{\partial \omega} = \sum_{i=1}^{N} \frac{\alpha}{\lambda(t_i)}(A(i) - \omega B(i)) - \alpha \sum_{i=1}^{n}(T - t_i)e^{-\omega(T-t_i)}$$

The expectation step remains as it is, but the maximization step for α and ω evolves to:

$$\alpha^{k+1} = \frac{\sum_{i>j} p_{ij}^k}{n - \sum_{i=1}^{n} e^{-\omega(T-t_i)}}$$

$$\omega^{k+1} = \frac{\sum_{i>j} p_{ij}^k}{\sum_{i>j}(t_i - t_j)p_{ij}^k + \alpha \sum_{i=1}^{n}(T - t_i)e^{-\omega^{k+1}(T-t_i)}}$$

Interestingly, the maximization step for ω^{k+1} is implicit. While we could solve this nonlinear equation at each iteration, we choose to use ω^k on the right side as a substitute for ω^{k+1}. One of the perks of this approach is that our algorithm maintains its nature as a projected gradient ascent [1]. The entries on the diagonal of $P(\Theta^k)$ are given by:

$$\mu^k/T,$$

$$\alpha^k/(n - \sum_{i=1}^{n} e^{-\omega(T-t_i)}),$$

$$\omega^k/(\sum_{i>j}(t_i - t_j)p_{ij}^k + \alpha \sum_{i=1}^{n}(T - t_i)e^{-\omega^{k+1}(T-t_i)}).$$

Expectation Maximization Algorithm

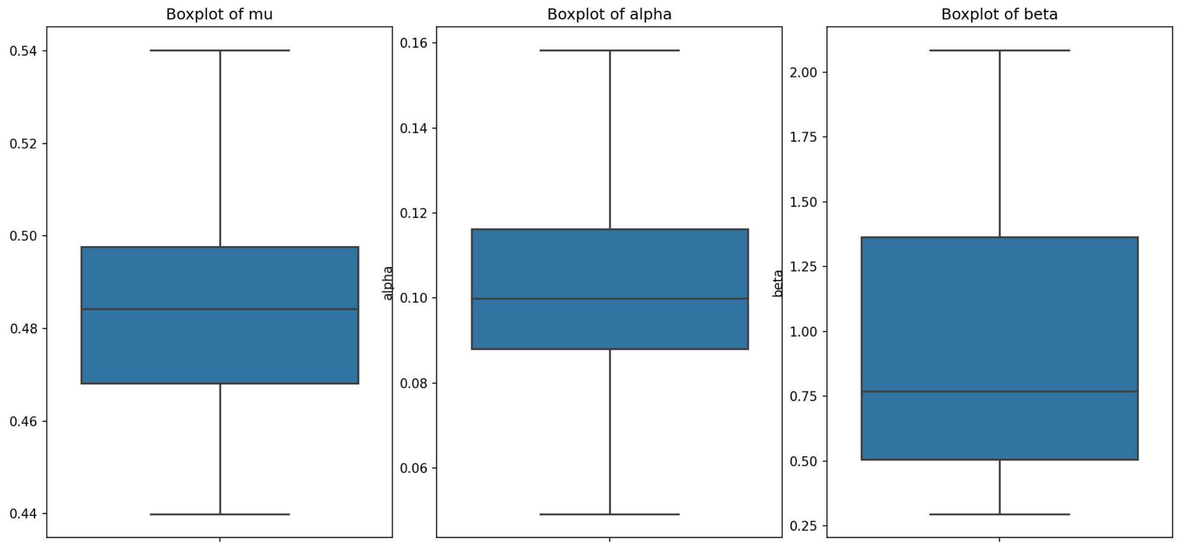
```
    void em_hawkes_process(std::vector<double>& times,
        double& alpha,
    double& beta, double& mu,
    int max_iterations = 100 , double tol = 1e-2) {
  int n = (int)( times.size());
  double diff;
  std::vector<double> pkij(n), pkii(n);
  for (int iter = 0; iter < max_iterations; ++iter)
  {
    double sum_pkij = 0, sum_pkii = 0, sum_weighted_pkij = 0;
    for (int i = 1; i < n; ++i)
    {
      double sum_exp = 0;
      for (int j = 0; j < i; ++j)
      {
        double temp = alpha * beta *
                    std::exp(-beta * (times[i] - times[j]));
        sum_exp += temp;
      }
      pkii[i] = mu / (mu + sum_exp);
      sum_pkii += pkii[i];

      for (int j = 0; j < i; ++j)
      {
        pkij[j] = alpha * beta *
                    std::exp(-beta*(times[i] - times[j]))/
                        (mu + sum_exp);
        sum_pkij += pkij[j];
        sum_weighted_pkij +=
            (times[i] - times[j]) * pkij[j];
      }
    }

    double sum_exp_wT_ti = 0;
    for (int i = 0; i < n; ++i) {
      sum_exp_wT_ti += std::exp(-beta *
                (times[n-1] - times[i]));
    }

    mu = sum_pkii / (times[n-1] - times[0]);
    alpha = sum_pkij / (n - sum_exp_wT_ti);
    beta = sum_pkij / sum_weighted_pkij;
  }
}
```

3.3.2 Estimation Results

If we simulate our path with the [18] Method with an interval from T = 0 to T = 2000 and with the parameters of above

$$\mu = 0.5 \quad \alpha = 0.1 \quad \beta = 1.5$$

we would like to obtain from our calibration approximatively the same calibrated paramaters.



We can see that the estimation is around the true values and therefore we can say that the estimation algorithm works correctly and we will be able to use it in the future with the true timestamp from our buckets.

3.3.3 Estimation with real trades timestamps

After having created our buckets of volume, we are going to check after how much time a bucket is filled and create a list of timestamp.
After that we are going to calibrate with the Expectation Maximization algorithm the timestamps and see which are the parameters for α, μ and β.

3.3.3.1 Estimation of the paramters

Calibrating with a bucket size of 1000 BTC for buckets we end up having that

$$\alpha = 1.49508$$

$$\beta = 0.821979$$

$$\mu = 1.01967$$

and plugging those parameters in the Hawkes process we obtain these results:

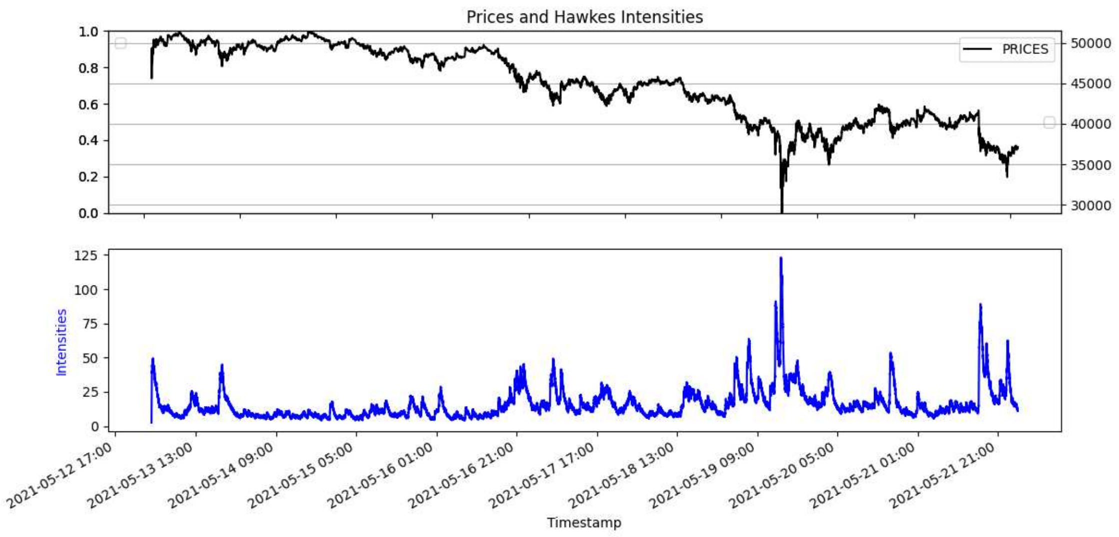


Figure 8: Price intensities

Upon observing the market dynamics, we find an interesting phenomenon that occurred around May 19, 2021. A substantial crash was observed during this time frame, an event which caught the attention of traders and researchers alike.

The unique feature of this particular incident was not just the downturn itself, but rather the subsequent events that unfolded in the market. Namely, there was a significant rise in the volume of Bitcoin exchanged.

Considering the nature of Hawkes processes, which are known to be self-exciting, this incident becomes even more noteworthy. This self-exciting process suggests a memory of past events, wherein each occurrence increases the probability of future events. It is a sort of snowball effect where the action doesn't die down immediately but has a persistent effect, leading to more events or actions. In the context of the Bitcoin market, this translates into an increase in trading volumes, with more transactions occurring in a shorter span of time.

Given that higher trading volume leads to more buckets in a shorter time frame, what we witnessed around May 19 was a drastic increase in our intensity.

3.4 Volume analysis

4 VPin

In this section we are going to introduce PIN in order to be able to speak about the main subject of this thesis which is the VPIN.

4.1 What is Pin

PIN a.k.a. Probability of informed trading is an indicator that assumes that in the market there are two different types of agents that enter in the market: the informed traders and uninformed traders.

4.1.1 Formula with MLE Approach

As cited in [9] given B_t and S_t the total number of daily buy and daily sell the joint likelihood function for a sample of buys and sells from $t = 1...T$ is given by:

$$\begin{aligned} L(\Theta|B_t, S_t) = \prod_{t=1}^{T} \Bigg[& \alpha\delta \frac{e^{-\mu+\lambda_s}(\mu+\lambda_b)^{S_t} e^{-\lambda_b(\lambda_b)^{B_t}}}{S_t! B_t!} + \\ & \alpha(1-\delta) \frac{e^{-(\mu+\lambda_b)^{B_t}} e^{-\lambda_s}(\lambda_s)^{S_t}}{S_t!} + \\ & (1-\alpha) \frac{e^{-\lambda_b}(\lambda_b)^{B_t} e^{-\lambda_s}(\lambda_s)^{S_t}}{B_t! S_t!} \Bigg] \end{aligned} \tag{2}$$

Where $\Theta = (\alpha, \delta, \mu, \lambda_b, \lambda_s)$.

Easley et al. (2002) estimated the vector of parameters Θ by maximizing the previous equation and the PIN formula turns out to be:

$$PIN = \frac{\alpha\mu}{\alpha\mu + \lambda_s + \lambda_b}$$

where $\alpha\mu + \lambda_s + \lambda_b$ is the arrival rate of all orders and $\alpha\mu$ is the arrival rate of informed orders.

4.2 What is Vpin

The Volume Synchronized Probability of Informed Trading (VPIN) is a mathematical model widely utilized in financial markets for various applications. It was originally introduced by Professors Maureen O'Hara and David Easley from Cornell University, in collaboration with Marcos Lopez de Prado from Tudor Investment Corporation and RCC at Harvard University

4.3 Introduction

Given a security with price S and present value S_0, after a certain amount of information is going to be incorporated into the price we are going to have that the price can be either S_B (Bad news) or S_G (Good news).

The probability that an event arrives within the time frame of the analysis is denoted as

α, and the probability that the news will be bad is δ (i.e. good $1-\delta$).

The author proved that the expected value of the security's price can be computed at time t as:

$\mathbb{E}[\mathbb{S}_{\approx}] = (1-\alpha_t)S_0 + \alpha_t[\delta_t S_B + (1-\delta_t)S_G])$.

Following a Poisson distribution with informed traders arriving with a rate μ, and uniformed traders with rate ϵ. Then in order to avoid losses from informed traders, market makers reach breakeven at a bid level

$$\mathbb{E}[B_t] = \mathbb{E}[S_t] - \frac{\mu\alpha_t(\delta_t)}{\epsilon + \mu\alpha_t(\delta_t)}(\mathbb{E}[S_t] - S_B).$$

and the breakeven ask level at time t must be

$$\mathbb{E}[A_t] = \mathbb{E}[S_t] + \frac{\mu\alpha_t(1-\delta_t)}{\epsilon + \mu\alpha_t(1-\delta_t)}(S_G - \mathbb{E}[S_t])$$

It follows that the breakeven ask-bid is given by:

$$\mathbb{E}[A_t - B_t] = \frac{\mu\alpha_t(1-\delta_t)}{\epsilon + \mu\alpha_t(1-\delta_t)}(S_G - \mathbb{E}[S_t]) + \frac{\mu\alpha_t(\delta_t)}{\epsilon + \mu\alpha_t(\delta_t)}(\mathbb{E}[S_t] - S_B).$$

if we consider the standard case we have

$$\delta = \frac{1}{2} \rightarrow \mathbb{E}[A_t - B_t] = \frac{\alpha_t\mu}{\alpha_t\mu + 2\epsilon}(S_G - S_B).$$

that gives us the information that the level at which market makers provide liquidity is

$$PIN_t = \frac{\alpha_t \mu}{\alpha_t \mu + 2\epsilon}$$

4.4 Low-Frequency Estimation

The **PIN** model needs estimation of four non-observable parameters, namely α, δ, μ and ϵ.

This was done via Maximum Likelihood, by fitting three Poisson distribution.

$$\mathbb{P}[V^B, V^S] = (1-\alpha)\mathbb{P}[V^B, \epsilon]\mathbb{P}[V^S, \epsilon] + \alpha(\delta\mathbb{P}[V^B, \epsilon]\mathbb{P}[V^S, \mu+\epsilon] + (1-\delta)\mathbb{P}[V^B, \mu+\epsilon]\mathbb{P}[V^S, \epsilon]).$$

Where V^B is the volume traded against the ask and V^S against the bid.

In [8] it has been proved that

$$\mathbb{E}[V^B - V^S] = \alpha(1-\delta)(\epsilon - (\mu+\epsilon)) + \alpha\delta(\mu + \epsilon - \epsilon) + (1-\alpha)(\epsilon - \epsilon) = \alpha\mu(1-2\delta)$$

and for sufficient large μ

$$\mathbb{E}[|V^S - V^B|] \approx \alpha\mu.$$

4.5 High-Frequency Estimation

David Easley, Marcos Lopez de Prado and Maureen O'Hara proposed a high-frequency estimate of PIN - **VPIN -** . The treshold is a volume clock captured by volume buckets. This is a form of subordinated stochastic process that differ from the standard chronological clock.

What has been done is divide samples of volume bars in volume buckets which are groups of trades that amount for the same aggregate volume.

Since all buckets have the same amount of volume we have

$$\frac{1}{n}\sum_{\tau=1}^{n}(V_{\tau}^{B}+V_{\tau}^{S})=V=\alpha\mu\epsilon$$

4.6 Results of VPIN

To calculate the VPIN, several parameters must be specified: the desired bucket size for analysis, the number of correlated buckets, and the filter threshold for orders deemed 'noisy'. For our analysis, we've set the bucket size at 1000 BTC, chosen 50 buckets for correlation, and established a \$100 threshold for filtering orders.

By running a simulation on a dataset, we obtained the following results:

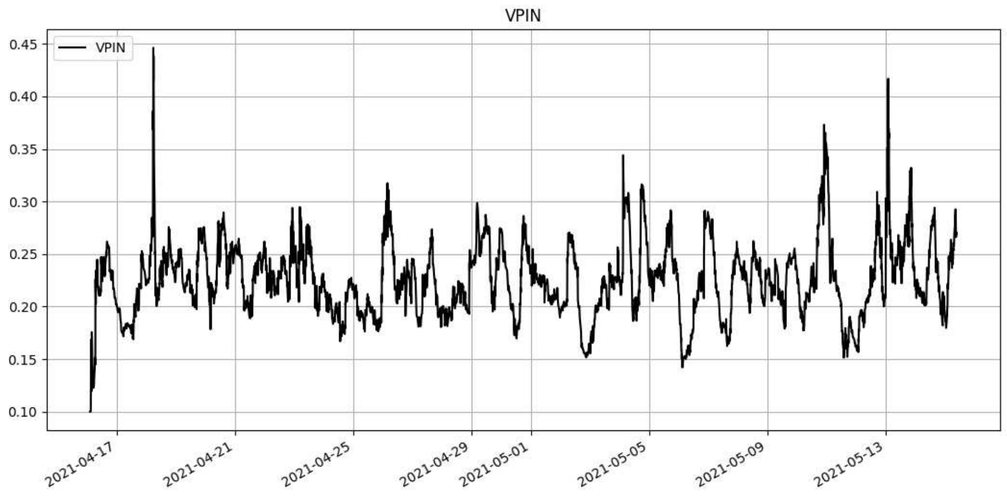


Figure 9: VPIN results

Having explored the preceding calculations (Order book depth, VPIN, Hawkes process), we now turn our attention to the most intriguing aspect of this thesis: determining whether the sign of returns can be predicted using the previously discussed properties. To achieve this, we will first delve into an explanation of the models we intend to employ.

5 Introduction to Machine Learning

As explained in the phd thesis [23]

Machine Learning (ML) is an interdisciplinary field combining computer science and statistics. It forms the crux of modern artificial intelligence (AI) by endowing computers with the ability to discern patterns and make informed decisions from data.

ML techniques are largely bifurcated into two primary paradigms:

1. **Supervised Learning -** Utilizes labeled data to prognosticate outcomes.
2. **Unsupervised Learning -** Endeavors to identify patterns in unlabeled data.

5.1 Supervised Learning

In supervised learning, data comes equipped with an answer key, and the algorithm iteratively adjusts itself to predict the correct output. The goal is to craft models that generalize patterns from given data in order to make predictions on new, unseen data. Notably, regression problems, the focus of this document, fall under this category.

5.2 Unsupervised Learning

Unsupervised learning differs from supervised learning in that there is no supervisor or absolute error measures. It involves algorithms independently discovering data structures within an unlabeled dataset $D = \{x_i\}_{i=1}^{N}$ containing feature vectors. The goal is to create a model that can transform these feature vectors into either other vectors or values useful for solving practical problems, uncovering "interesting patterns" in the data—a process often referred to as knowledge discovery. Unlike supervised learning, there are no predefined patterns to search for, and no clear error metric for evaluation.

5.3 Our Focus

What we are more interested about in this thesis is supervised learning.

We are going then to cover most of the arguments that are related to this in the next pages from.

5.3.1 Linear Regression

Regression serves as a foundational technique widely applied in data analysis, primarily focusing on two pivotal objectives: data prediction and forecasting. These areas find significant relevance within the domain of machine learning, making regression an indispensable tool for projects in this field. Furthermore, regression analysis extends its utility into areas characterized by the exploration of causal relationships among both dependent and independent variables within datasets featuring diverse variables. Across all models of regression, a fundamental principle underpins their operation: leveraging independent variables to forecast dependent variables [24].

Within the landscape of regression analysis, linear regression stands as a distinctive model, characterized by its exclusive reliance on a single independent variable to ascertain dependent variables [2]. These models play a pivotal role in assessing the influence exerted by independent variables on designated dependent variables. In the process of model development, linear regression adheres to a straightforward linear approach, aiming to minimize disparities between predicted values and actual outputs. Employing a linear regression model facilitates the estimation of data, drawing upon actual observations drawn from a random sample of data. Additionally, linear regression assists in identifying the presence or absence of a linear relationship between input and output variables. When applying linear regression to a dataset, the model rigorously selects a target variable under specific predefined conditions while excluding extraneous data points. This meticulous approach not only results in the creation of a functional model but also streamlines the grouping of data with shared characteristics, enhancing organizational clarity.

5.3.2 Logistic Regression

Logistic regression stands as one of the key machine learning algorithms specifically tailored for assessing dependent variables when they exhibit binary outcomes. When applied to datasets of this nature, logistic regression facilitates the development of predictive models capable of discerning appropriate outcomes. Utilizing this algorithm not only enables data description but also the elucidation of relationships between a dependent variable and other nominal, ordinal, or interval variables [21].

Logistic regression encompasses various types that are chosen based on the inherent characteristics of the data, each suited for specific classification and prediction tasks. Common variants of logistic regression analysis include binary logistic regression, multinomial logistic regression, and ordinal logistic regression.

In the realm of binary logistic regression, there exist only two possible categorical responses, and all data points are categorized into one of these two potential values. On the other hand, multinomial logistic regression handles datasets with more than two categories, assigning each data point to one of the available classes. These classes are typically unordered, and their selection is driven by feature matching.

Finally, ordinal logistic regression tackles datasets featuring multiple ordered categories. In this scenario, incoming data is assigned to the appropriate category based on feature matching. This comprehensive approach ensures that logistic regression can effectively handle diverse datasets and address various classification challenges.

hence, the majority of the situations that will be coming in the real life situations will be addressed by the regression analysis, and hence the mode training can be done effectively.

Depending on the nature of the dataset, a more appropriate variant of logistic regression will be selected for the model's development.

5.3.3 Ordinary Least Square

In our analysis we used first a simple OLS[19] method as baseline to compare later with a Neural Network the results.

In statistics, **ordinary least squares** (**OLS**) is a type of *linear least squares* method for choosing the unknown parameters in a *linear regression* model (with fixed level-one effects of a linear function of a set of explanatory variables) by the principle of least squares: minimizing the sum of the squares of the differences between the observed dependent variable (values of the variable being observed) in the input dataset and the output of the (linear) function of the independent variable.

Geometrically, this is seen as the sum of the squared distances, parallel to the axis of the dependent variable, between each data point in the set and the corresponding point on the regression surface—the smaller the differences, the better the model fits the data. The resulting estimator can be expressed by a simple formula, especially in the case of a simple linear regression, in which there is a single regressor on the right side of the regression equation.

The OLS estimator is consistent for the level-one fixed effects when the regressors are exogenous and forms perfect collinearity (rank condition), consistent for the variance estimate of the residuals when regressors have finite fourth moments and—by the Gauss–Markov theorem—optimal in the class of linear unbiased estimators when the errors are homoscedastic and serially uncorrelated. Under these conditions, the method of

OLS provides minimum-variance mean-unbiased estimation when the errors have finite variances. Under the additional assumption that the errors are normally distributed with zero mean, OLS is the maximum likelihood estimator that outperforms any non-linear unbiased estimator.

5.4 Linear Model

Assume the dataset comprises n samples $\{\mathbf{z}_i, w_i\}_{i=1}^{n}$. For every sample i, there is a scalar outcome w_i and a column vector $\mathbf{z}_i$ with p features, i.e., $\mathbf{z}_i = [z_{i1}, z_{i2}, \ldots, z_{ip}]^{\mathsf{T}}$. In the linear regression framework, the outcome, w_i, is represented as a linear combination of the features:

$$w_i = \alpha_1 \ z_{i1} + \alpha_2 \ z_{i2} + \cdots + \alpha_p \ z_{ip} + \delta_i,$$

or in its vector notation,

$$w_i = \mathbf{z}_i^{\mathsf{T}} \boldsymbol{\alpha} + \delta_i,$$

$$\mathbf{w} = \mathbf{Z}\boldsymbol{\alpha} + \boldsymbol{\delta},$$

For an overdetermined system, we have

$$\sum_{j=1}^{p} z_{ij}\alpha_j = w_i,\ (i = 1, 2, \dots, n),$$

$$\mathbf{Z}\boldsymbol{\alpha} = \mathbf{w},$$

$$\mathbf{Z} = \begin{bmatrix} Z_{11} & Z_{12} & \cdots & Z_{1p} \\ Z_{21} & Z_{22} & \cdots & Z_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{n1} & Z_{n2} & \cdots & Z_{np} \end{bmatrix}, \qquad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix}, \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}.$$

$$\hat{\boldsymbol{\alpha}} = \underset{\boldsymbol{\alpha}}{\arg\min}\, L(\boldsymbol{\alpha}),$$

Where the objective function $L(\boldsymbol{\alpha})$ is given by:

$$L(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \left| w_i - \sum_{j=1}^{p} Z_{ij}\alpha_j \right|^2 = \left\| \mathbf{w} - \mathbf{Z}\boldsymbol{\alpha} \right\|^2.$$

$$\left(\mathbf{Z}^{\mathsf{T}}\mathbf{Z}\right)\hat{\boldsymbol{\alpha}} = \mathbf{Z}^{\mathsf{T}}\mathbf{w}\ .$$

The matrix $\left(\mathbf{Z}^{\mathsf{T}}\mathbf{Z}\right)$ is known as the normal matrix and the $\mathbf{Z}^{\mathsf{T}}\mathbf{w}$ as the moment matrix. Finally, $\hat{\boldsymbol{\alpha}}$ is the coefficient vector the l-s hyperplane expressed as:

$$\hat{\boldsymbol{\alpha}} = \left(\mathbf{Z}^{\mathsf{T}}\mathbf{Z}\right)^{-1}\mathbf{Z}^{\mathsf{T}}\mathbf{w}.$$

or

$$\hat{\boldsymbol{\alpha}} = \boldsymbol{\alpha} + \left(\mathbf{Z}^{\mathsf{T}}\mathbf{Z}\right)^{-1}\mathbf{Z}^{\mathsf{T}}\boldsymbol{\delta}.$$

Consider that b is a potential choice for the parameter vector β. The term $y_i - x_i^T b$, known as the *deviation* for the i-th entry, denotes the vertical gap between the data point (x_i, y_i) and the hyperplane $y = x^T b$. This evaluates how well the model fits the data. The *sum of squared deviations* (SSD), also referred to as the *deviation sum of squares* (DSS) or *residual sum of squares* (RSS) ([12], pg.15) , quantifies the comprehensive model fit:

$$S(b) = \sum_{i=1}^{n}(y_i - x_i^T b)^2 = (y - Xb)^T(y - Xb),$$

where T stands for the matrix transpose. The rows of X, representing the values of all independent variables linked with a particular dependent variable value, are $X_i = x_i^T$. The b value that minimizes this sum is termed the **OLS estimate for** β. $S(b)$ is a quadratic function in b with a positive-definite Hessian, ensuring this function has a unique global minimum at $b = \hat{\beta}$, defined by:

$$\hat{\beta} = \arg\min_{b \in \mathbb{R}^p} S(b) = (X^T X)^{-1} X^T y.$$

Here, $N = X^T X$ is a Gram matrix, and its inverse, $Q = N^{-1}$, is the *influence matrix* of β [10] [13]([32], pg.134). The product $X^T X^{-1} X^T = Q X^T$ is named the Moore–Penrose pseudoinverse matrix of X. This elucidation underscores that estimations are valid only if there's no absolute multicollinearity among the explanatory variables.

After determining β, the *approximated values* from the regression are

$$\hat{y} = X\hat{\beta} = Py,$$

with $P = X(X^T X)^{-1} X^T$ being the *projection matrix* onto space V covered by X's columns. This matrix P is also dubbed the *hat matrix* as it "adorns" the variable y. Another matrix, $M = I_n - P$, known as the *nullifier* matrix, projects onto the space orthogonal to V. Both P and M are symmetric and idempotent, relating to X via $PX = X$ and $MX = 0$ ([12], pg.15). Matrix M yields the *deviations* from the regression:

$$\hat{\varepsilon} = y - \hat{y} = y - X\hat{\beta} = My = M(X\beta + \varepsilon) = (MX)\beta + M\varepsilon = M\varepsilon.$$

Using these deviations, σ^2's value can be approximated via the *reduced chi-squared* statistic:

$$s^2 = \frac{\hat{\varepsilon}^T \hat{\varepsilon}}{n-p} = \frac{y^T M^T M y}{n-p} = \frac{y^T M y}{n-p} = \frac{S(\hat{\beta})}{n-p},$$

$$\hat{\sigma}^2 = \frac{n-p}{n} s^2.$$

The bottom value, $n-p$, represents the statistical degrees of freedom. The initial value, s^2, is the OLS approximation for σ^2, whereas $\hat{\sigma}^2$ is known to be an unbiased approximation of σ^2([14], pg.187)([34], pg.626). These two concepts are elucidated for researchers acquainted with finite sample theory, a significant aspect of multivariate statistics.

A common method to evaluate the OLS regression's fit is by measuring how much the original data variation can be lessened by regressing on X. The *determination coefficient* R^2 is the ratio of the "explained" variance to the "total" variance of y, when the regression sum equals the squared residuals' sum ([12], pg.15):

$$R^2 = \frac{\sum(\hat{y}_i - \overline{y})^2}{\sum(y_i - \overline{y})^2} = \frac{y^{\mathrm{T}} P^{\mathrm{T}} L P y}{y^{\mathrm{T}} L y} = 1 - \frac{y^{\mathrm{T}} M y}{y^{\mathrm{T}} L y} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} \tag{3}$$

where TSS is the dependent variable's *overall sum of squares*, $L = I_n - \frac{1}{n} J_n$, and J_n is an all-ones $n \times n$ matrix. L centralizes a variable by subtracting its mean. R^2 is meaningful if the data matrix X has a ones column, indicating the coefficient is the regression intercept. Then, R^2 lies between 0 and 1, with values nearing 1 signifying a close fit.

The variance in predicting the independent variable based on the dependent variable is discussed in the article titled *Polynomial least squares.*

6 Neural Networks

6.1 What is a Neural Network

A Neural Network is a complex structure comprising layers stacked on top of each Nother, with neurons residing within each layer. It serves as a mathematical framework for mapping input variables to a desired target variable and has the capability to acquire and recognize patterns.

6.2 Architecture of a NN

[27]Let's describe the structure of a Neural Network. For the sake of illustration, we will consider a 2-layer Neural Network.

The input layer receives two input features, denoted as x_1 and x_2. In the hidden layer, there are three neurons, each of which is associated with weight parameters :

$(w_{11}, w_{12}, w_{13}, w_{21}, w_{22}, w_{23}, w_{31}, w_{41}, w_{51})$, as depicted in the figure below [Figure 8]. Typically, these weight parameters are initialized with random values.

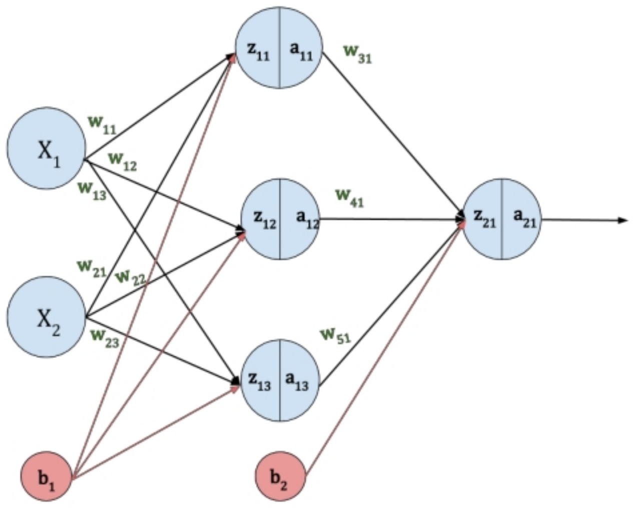


Figure 10: Simple Neural Network

`https://studymachinelearning.com/mathematics-behind-the-neural-network/`

The terms b_1 and b_2 represent bias parameters corresponding to the input layer and hidden layer, respectively. The Neural Network encompasses both Forward Propagation and Back-propagation, involving a sequence of five essential steps:

1. Initialization of weight and bias parameters.
2. Execution of Forward Propagation.
3. Calculation of the Loss.
4. Implementation of Back-propagation.
5. Adjustment of the weight and bias parameters.

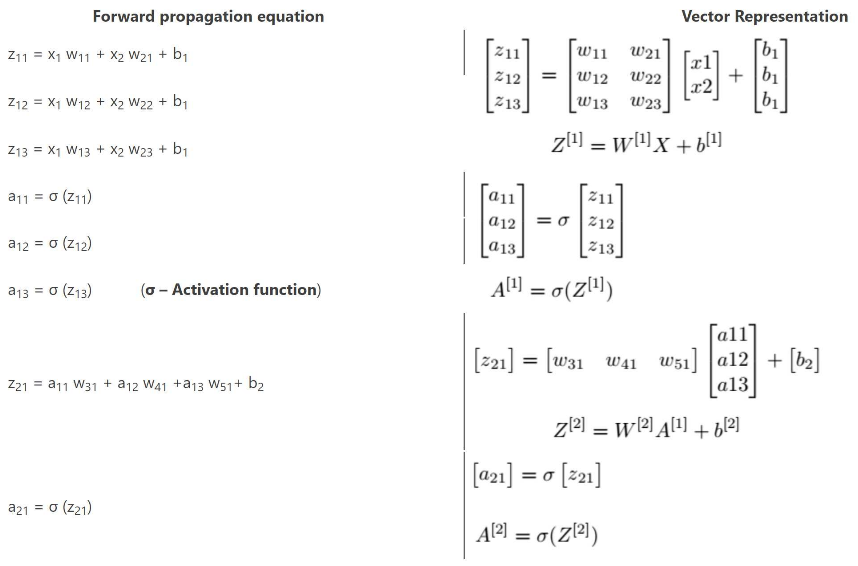
6.2.1 Forward Propagation

[27]During Forward Propagation, input data is sequentially processed through the network in a forward manner. Each hidden layer undertakes data processing, performs computations, and transfers results to the subsequent layer. Ultimately, the output layer computes the model's output.

Now, let's delve into the mathematical expressions that define Forward Propagation. z_{11}, z_{12}, z_{13}, and z_{21} are intermediate neuron values derived from the weights, biases, and neuron values of the preceding layer.

An activation function is subsequently applied to these intermediate neuron values to capture non-linear patterns between the input and the target output variable. As a result, a_{11}, a_{12}, a_{13}, and a_{21} represent the outputs of the activation function applied to z_{11}, z_{12}, z_{13}, and z_{21}, respectively.

Below we can see a table with the representation of the equations and the vector representation of them.



Forward propagation equation	**Vector Representation**
$z_{11} = x_1 w_{11} + x_2 w_{21} + b_1$ $z_{12} = x_1 w_{12} + x_2 w_{22} + b_1$ $z_{13} = x_1 w_{13} + x_2 w_{23} + b_1$	$\begin{bmatrix} z_{11} \\ z_{12} \\ z_{13} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \begin{bmatrix} x1 \\ x2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_1 \\ b_1 \end{bmatrix}$ $Z^{[1]} = W^{[1]}X + b^{[1]}$
$a_{11} = \sigma(z_{11})$ $a_{12} = \sigma(z_{12})$ $a_{13} = \sigma(z_{13})$ (**σ – Activation function**)	$\begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} = \sigma \begin{bmatrix} z_{11} \\ z_{12} \\ z_{13} \end{bmatrix}$ $A^{[1]} = \sigma(Z^{[1]})$
$z_{21} = a_{11} w_{31} + a_{12} w_{41} + a_{13} w_{51} + b_2$	$\begin{bmatrix} z_{21} \end{bmatrix} = \begin{bmatrix} w_{31} & w_{41} & w_{51} \end{bmatrix} \begin{bmatrix} a11 \\ a12 \\ a13 \end{bmatrix} + \begin{bmatrix} b_2 \end{bmatrix}$ $Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$
$a_{21} = \sigma(z_{21})$	$\begin{bmatrix} a_{21} \end{bmatrix} = \sigma \begin{bmatrix} z_{21} \end{bmatrix}$ $A^{[2]} = \sigma(Z^{[2]})$

Figure 11: Formulas for forward propagation

`https://studymachinelearning.com/mathematics-behind-the-neural-network/`

6.2.2 Common Loss Functions

The Mean Squared Error[20]:

Given a set of n predictions derived from a sample of n data points involving multiple variables, and Y represents the vector of observed values for the predicted variable, while $\hat{Y}$ corresponds to the predicted values (for instance, obtained through a least-squares fit), you can calculate the Mean Squared Error (MSE) within the sample as follows:

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i \right)^2 .$$

Cross Entropy[6]:

Cross-entropy loss, also known as log loss, evaluates the effectiveness of a classification model that produces probability values ranging from 0 to 1. As the predicted probability deviates further from the true label, the cross-entropy loss escalates.

For binary classification ($M = 2$), the cross-entropy loss is calculated as:

$$-[y \log(p) + (1 - y) \log(1 - p)]$$

6.2.3 Activation Function

In place of the conventional logistic sigmoid function, we can consider the hyperbolic tangent function, denoted as $\tanh(z)$:

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

As stated in [26] Similar to the logistic sigmoid, the tanh function also exhibits a sigmoidal (”S”-shaped) behavior. However, it yields values within the range of $(-1, 1)$. Consequently, strongly negative inputs to the tanh function will be mapped to negative outputs. Furthermore, only zero-valued inputs result in nearly zero outputs. These characteristics make it less likely for the network to become ”stuck” during training.We know that the gradient for the tanh function is:

$$\frac{d}{dz}\tanh(z) = \frac{\partial}{\partial z}\left(\frac{\sinh(z)}{\cosh(z)}\right) = 1 - \tanh^2(z)$$

6.2.4 Backword Propagation

[27]Backward propagation, often referred to as backpropagation, is the process of transmitting the error (loss) back through the neural network and subsequently updating the weights and bias parameters.

Backpropagation plays a pivotal role in the operation of Neural Networks, executing various mathematical operations to discern patterns between the input and the target variable.

The primary objective of the neural network is to minimize the error (loss). Achieving this involves obtaining accurate values for the weight and bias parameters.

The error undergoes changes concerning these parameters, and the rate of this change in error is determined by computing the partial derivatives of the loss function with respect to each parameter.

Through these derivatives, we ascertain the sensitivity of the loss function to each weight and bias parameter. This approach is commonly known as the Gradient Descent[31] optimization method.

Let's delve into how backpropagation functions with an illustrative example. In this context, we need to calculate the partial derivatives of the loss function with respect to each parameter. The subsequent section outlines the computations involved in this derivation.

In the diagram above, the red line signifies the backpropagation mechanism. The terms

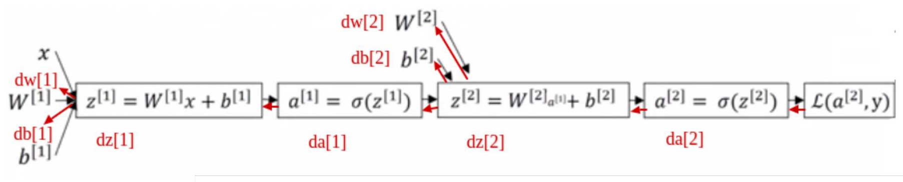

dw[2] $W^{[2]}$
db[2] $b^{[2]}$
x
dw[1]
$W^{[1]}$
db[1]
$b^{[1]}$
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$\mathcal{L}(a^{[2]}, y)$
dz[1]
da[1]
dz[2]
da[2]

Figure 12: Formulas for backward propagation

`https://studymachinelearning.com/mathematics-behind-the-neural-network/`

da[2], dz[2], dw[2], db[2], da[1], dz[1], dw[1], and db[1] denote the partial derivatives of the loss function concerning a[2], z[2], w[2], b[2], a[1], z[1], w[1], and b[1], respectively. While computing these derivatives, it is also imperative to calculate the derivative of the activation function.

Now let's sum up these derivative for the cross-entropy case [27]:

$$da^{[2]} = \frac{\partial L}{\partial a^{[2]}} =$$

$$= \frac{\partial L}{\partial a^{[2]}} \left(-y \log(a^{[2]} - (1-y) \log(1 - a^{[2]} \right) =$$

$$= \frac{-y}{a^{[2]}} + \frac{(1-y)}{1 - a^{[2]}}$$

$$dz^{[2]} = \frac{\partial L}{\partial z^{[2]}} = \frac{\partial L}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} =$$

$$= \frac{-y}{a^{[2]}} + \frac{(1-y)}{1 - a^{[2]}} \times \frac{\partial \sigma(z^{[2]})}{\partial z^{[2]}} =$$

$$= \frac{-y(1 - a^{[2]}) + (1-y)a^{[2]}}{a^{[2]}(1 - a^{[2]})} \times a^{[2]}(1 - a^{[2]}) =$$

$$= -y + ya^{[2]} + a^{[2]} - ya^{[2]} = a^{[2]} - y$$

with $a^{[2]} = \sigma(z^{[2]})$

$$dw^{[2]} = \frac{\partial L}{\partial w^{[2]}} =$$

$$= \frac{\partial L}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial w^{[2]}} =$$

$$= dz^{[2]} \times \frac{\partial (w^{[2]} a^{[1]} + b^{[2]})}{\partial w^{[2]}} = dz^{[2]} \times a^{[1]}$$

$$db^{[2]} = \frac{\partial L}{\partial b^{[2]}} =$$

$$= \frac{\partial L}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial b^{[2]}} =$$

$$= dz^{[2]} \times \frac{\partial (w^{[2]} a^{[1]} + b^{[2]})}{\partial b^{[2]}} =$$

$$= dz^{[2]}$$

$$da^{[1]} = \frac{\partial L}{\partial a^{[1]}} =$$

$$= \frac{\partial L}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} =$$

$$= dz^{[2]} \times \frac{\partial (w^{[2]} a^{[1]} + b^{[2]})}{\partial a^{[1]}} =$$

$$= dz^{[2]} \times w^{[2]}$$

$$dz^{[1]} = \frac{\partial L}{\partial z^{[1]}} =$$

$$= \frac{\partial L}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} =$$

$$= (dz^{[2]} \times w^{[2]}) \times \frac{\sigma(z^{[1]})}{\partial z^{[1]}} =$$

$$= (dz^{[2]} \times w^{[2]}) \times \sigma'(z^{[1]})$$

$$dw^{[1]} = \frac{\partial L}{\partial w^{[1]}} = \frac{\partial L}{\partial z^{[1]}} \times \frac{\partial z^{[1]}}{\partial w^{[1]}} =$$

$$= dz^{[1]} \times \frac{\partial (w^{[1]}x + b^{[1]})}{\partial w^{[1]}} =$$

$$= dz^{[1]} \times x$$

$$db^{[1]} = \frac{\partial L}{\partial z^{[1]}} = \frac{\partial L}{\partial z^{[1]}} \times \frac{\partial z^{[1]}}{\partial w^{[1]}} =$$

$$= dz^{[1]} \times \frac{\partial (w^{[1]}x + b^{[1]})}{\partial w^{[1]}} =$$

$$= dz^{[1]} \times x$$

6.2.4.1 Weight Update

The parameters for weight and bias are revised by deducting the gradient of the loss function with respect to these parameters.

The learning rate, denoted by α, dictates the magnitude of the parameter update. It determines the extent of adjustment to the parameter during each update step. The value of α ranges from 0 to 1, allowing for controlled modifications to the model's parameters.

$$W^{[1]} = W^{[1]} - \alpha \frac{\partial L}{\partial W^{[1]}} \qquad W^{[2]} = W^{[2]} - \alpha \frac{\partial L}{\partial W^{[2]}}$$

$$b^{[1]} = b^{[1]} - \alpha \frac{\partial L}{\partial b^{[1]}} \qquad b^{[2]} = b^{[2]} - \alpha \frac{\partial L}{\partial b^{[2]}}$$

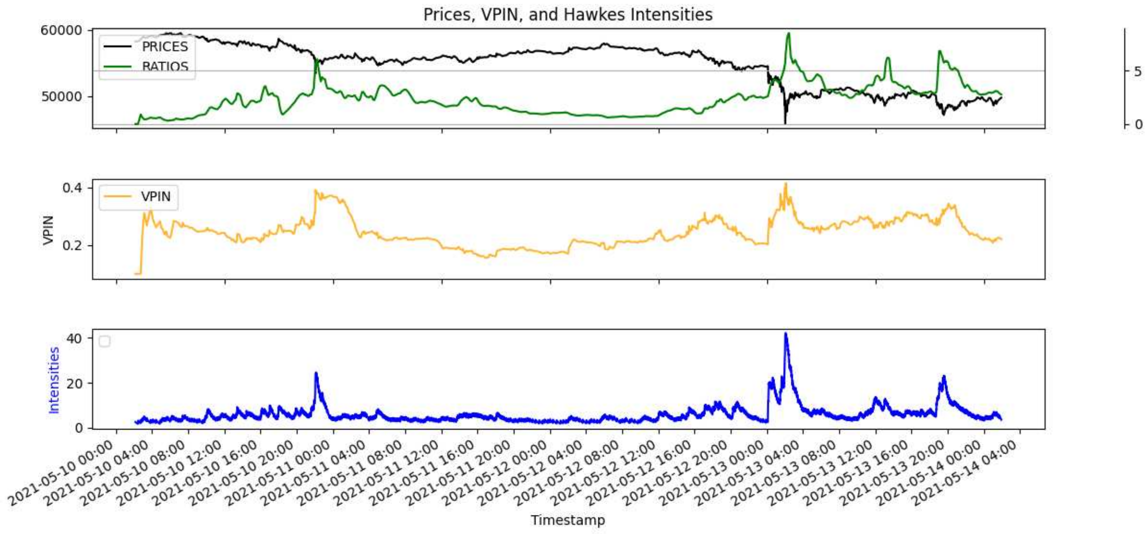
7 Backtests

Before we delve into the intricacies of the backtest, it's essential to familiarize ourselves with the behavior and values of our chosen indicators, especially during the initial phases. For this purpose, we shall first inspect how these indicators performed during the very first 4 days. Following that, we will broaden our horizon and observe them over an extended timeframe, particularly during periods characterized by significant market fluctuations.

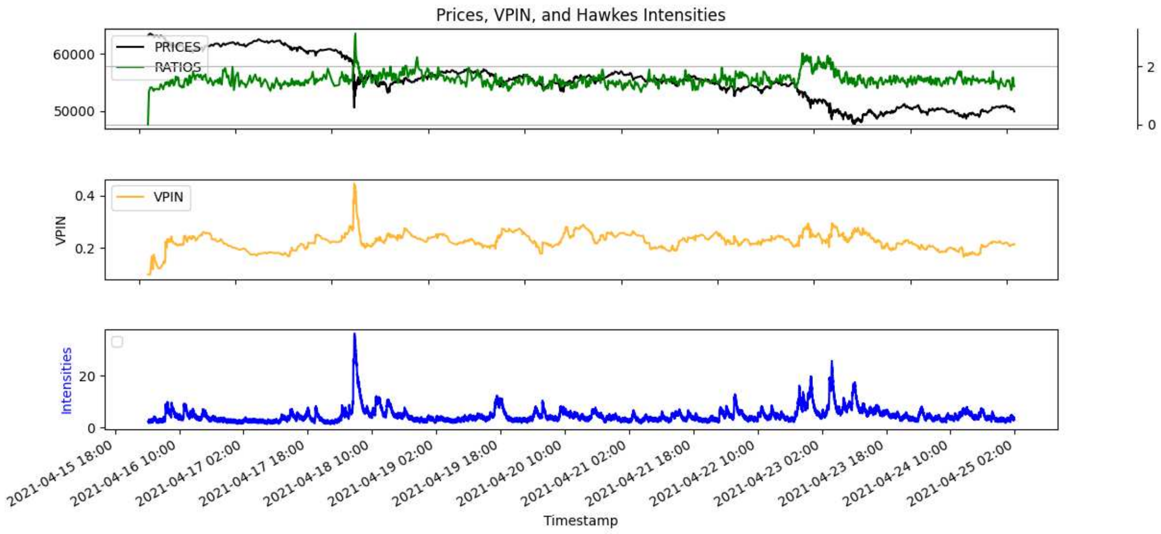
The graphs that we will be presenting next will showcase three main components:

- The market prices
- The values of vpin
- The hawkes intensities

Now, let's turn our attention to the first visualization. The following graph represents the results and values of our indicators during the initial 4 days:



However, to gain a deeper understanding, it's also invaluable to study these indicators during tumultuous periods in the market. The subsequent graph encapsulates the data from a phase where the market experienced a pronounced crash:



In the realm of financial modeling and analysis, it's vital to ensure that methodologies are not only robust but also meticulously detailed to ensure reliable outcomes. Here's an in-depth look into the comprehensive method I adopted for my backtest:

1. **Bucketing Process**: The initial step involved creating a series of buckets. Each bucket represented a specific volume of traded assets. As trades occurred and the specified volume threshold of a bucket was reached, I recorded pertinent data: chiefly the price at which the trade occurred, along with several other relevant features.

2. **Lagged Returns Calculation**: With buckets established and data recorded, I proceeded to compute the lagged returns of these buckets. This step was critical as it provided a historical perspective, allowing us to see how returns behaved in previous periods relative to specific volumes and other features.

3. **Feature Selection**: Four primary features emerged as particularly crucial for predictive modeling. These were:

    - Lagged returns from the aforementioned process.
    - VPIN, offering insights into volume-synchronized probability of informed trading.
    - Order Book Imbalance, which provides a snapshot of the discrepancy between buying and selling pressures.

- Hawkes Intensity, capturing the self-exciting dynamics of certain financial events.

4. **Model Training**: Armed with these four features, I embarked on training two distinct models: a Neural Network (NN) and an Ordinary Least Squares (OLS) model. The data extracted from the buckets served as the foundational training set for these models.

5. **Train-Test Split**: Ensuring a model's effectiveness isn't complete without out-of-sample testing. I partitioned the data into training and testing subsets using the classic train-test division methodology. This split guaranteed that the model's performance wasn't merely a result of overfitting to the training data.

6. **Result Analysis**: Post-training, I tested the models using the out-of-sample data. The results yielded were returns in percentage terms. Any positive return ($> 0\%$) was coded as '1' and any negative return ($< 0\%$) as '-1'.

7. **Accuracy Assessment**: The final step involved assessing the accuracy of the model's predictions. This was achieved by calculating the ratio of instances where the predicted sign of return matched the actual sign of return to the total number of predictions. The formula used was:

$$\text{Accuracy} = \frac{\text{Total of Correct Sign Predictions}}{\text{Total of All Predicted Signs}}$$

This meticulous process, which seamlessly integrated data bucketing with advanced predictive modeling techniques, was pivotal in ensuring reliable and actionable insights.

In our pursuit to validate and illustrate the performance of both the Neural Network (NN) and the Ordinary Least Squares (OLS) model, we have prepared a series of graphical representations. These are designed not only to provide quantitative insights but also to offer a visual assessment of the models' predictions relative to actual outcomes. Here's a detailed breakdown of the visualizations:

1. **Returns Comparison**:

   This graph juxtaposes the predicted returns with the true returns. For both the NN and the OLS model, this visualization serves as an immediate gauge of how accurately our models have forecasted compared to the actual data.

2. **Visual Verification of Returns and Prices**:

   Through a color-coded scheme, this graph presents the return predictions alongside the actual prices. The purpose of this is twofold: to visually confirm the direction of the returns (positive or negative) and to correlate these predictions with price movements. This method offers a more intuitive way to spot-check the correctness of our model predictions.

7.0.1 General Metrics

1. **Accuracy**: This metric provides us with the ratio of correct predictions (positive return or negative return) to the total number of input samples. It's a primary measure to understand how well our model is performing. Mathematically, the formula for accuracy is given by:

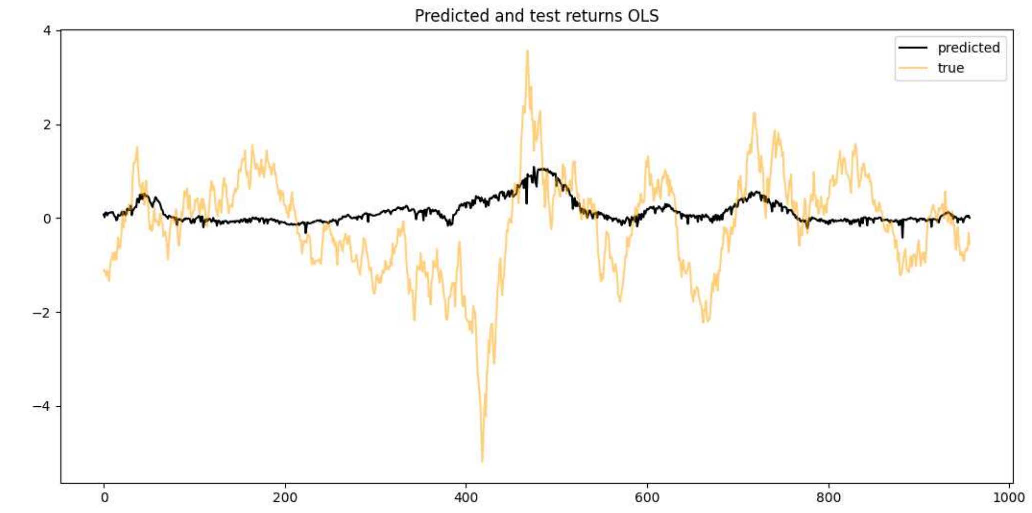
$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100$$

2. **Mean Absolute Error (MAE)**: This represents the average of the absolute differences between the predicted and actual values. It gives a snapshot of the magnitude of errors made by our model.

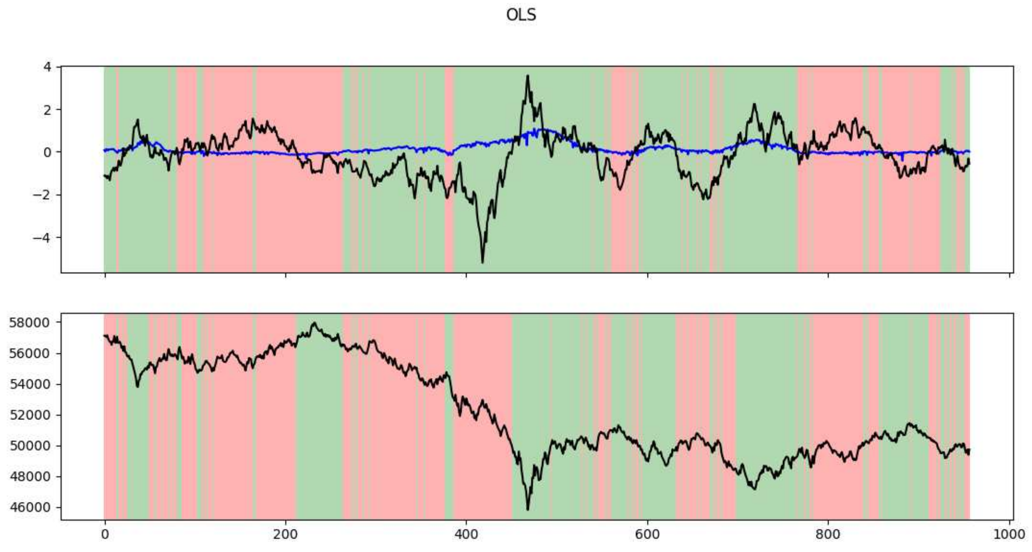
3. **Mean Squared Error (MSE [20])**: This metric takes the average of the squared differences between the predicted and actual values. It provides insights into the variance of our predictions and is especially useful when we want to penalize larger errors more.

7.1 OLS Result

1. **Returns Comparison**:



2. **Visual Verification of Returns and Prices**:



7.1.1 OLS metrics and parameters

Intercept: [0.05514604]

Coefficients: [0.00640321 0.17465479 -0.03895252 -0.03441836]

Accuracy 46.29 %

Correct 443

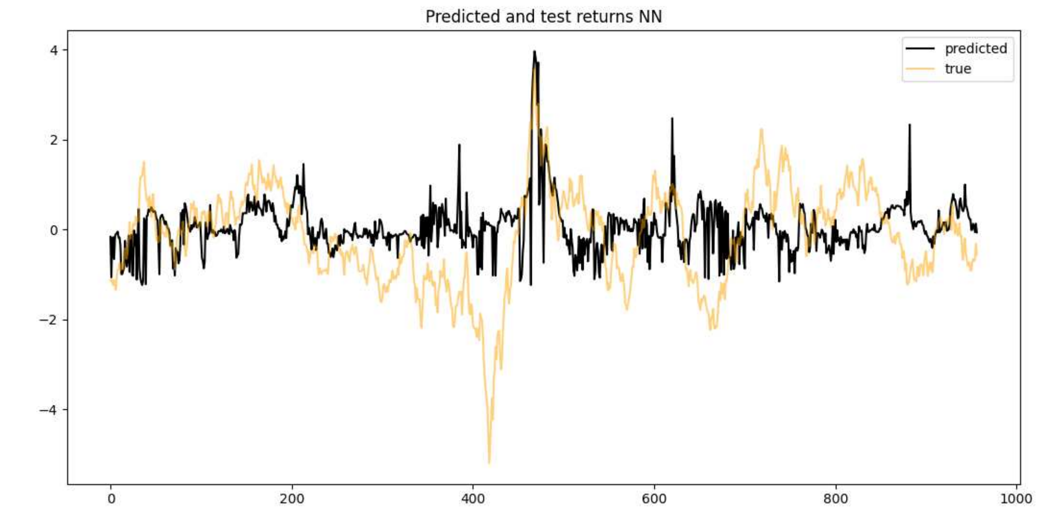
Number of elements 957

Mean Absolute Error: 0.8290

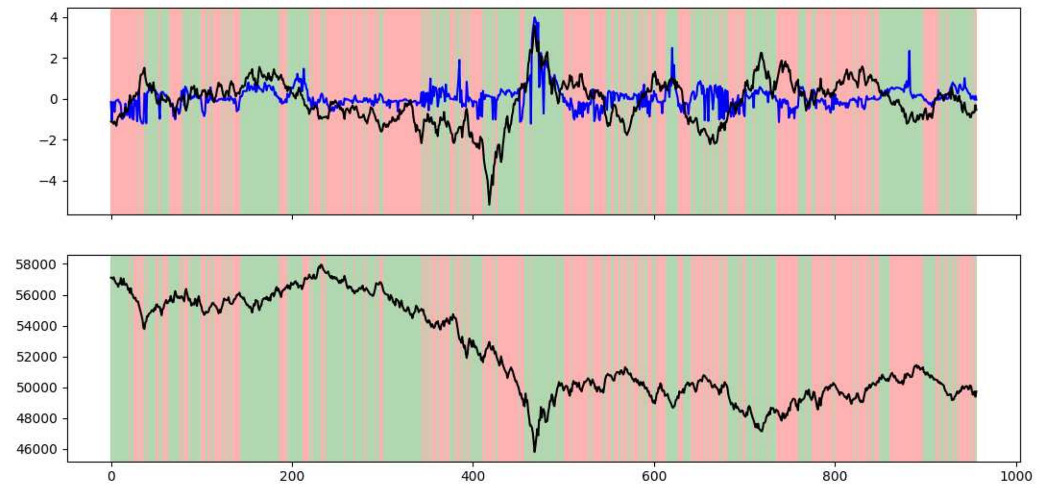
Mean Squared Error: 1.2128

7.2 NN Result

1. **Returns Comparison**:



2. **Visual Verification of Returns and Prices**:



7.2.1 Neural Network metrics

It's essential to acknowledge that these interpretations are derived from isolated weights, and the overall behavior of the model is shaped by the interplay of all features, the chosen activation functions, and biases (which haven't been provided). Additionally, domain-specific insights from financial time series analysis can further refine these interpretations.

Accuracy 52.77 %

Correct 505

Number of elements 957

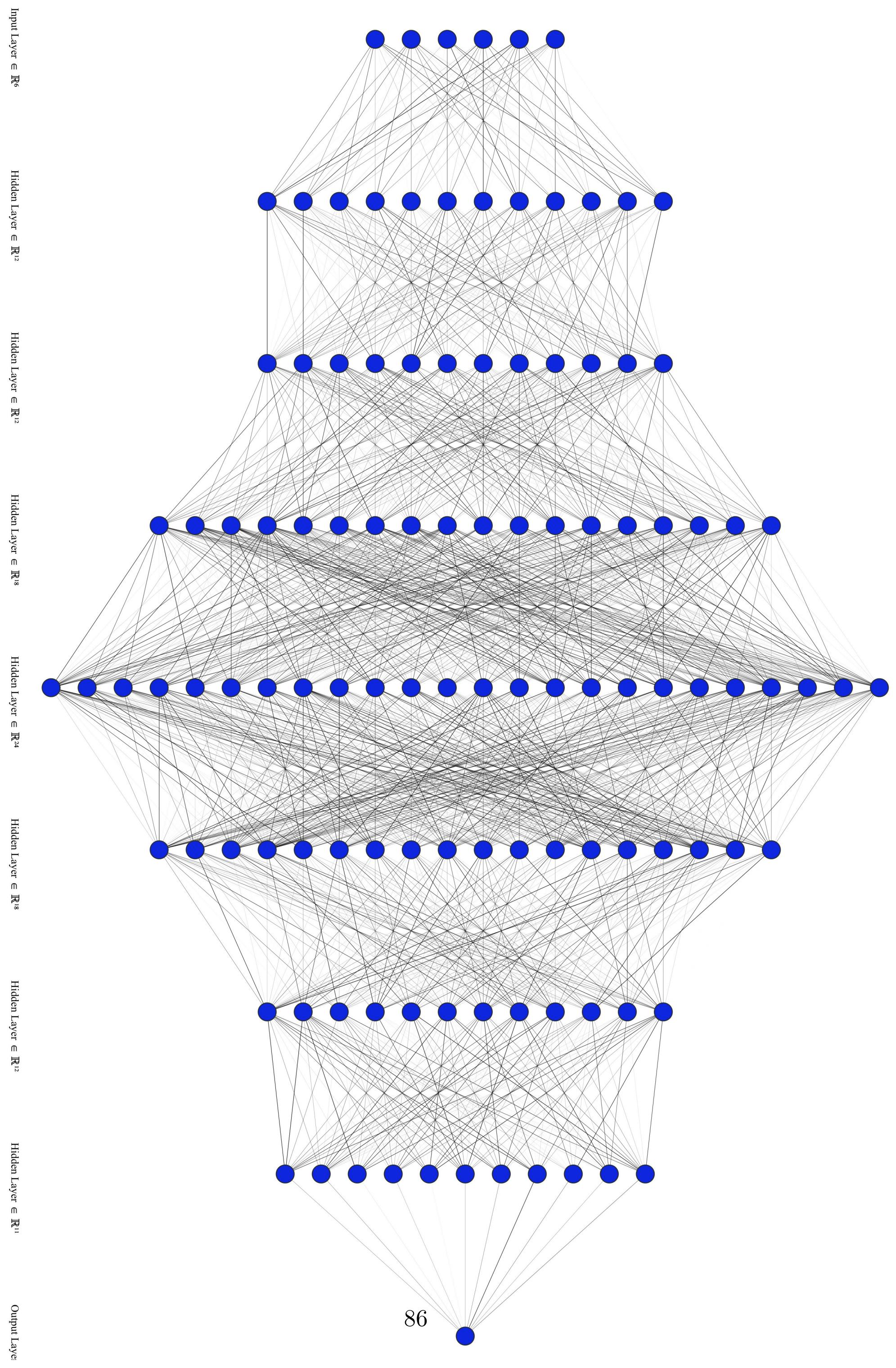
Mean Absolute Error: 0.8843

Mean Squared Error: 1.3017

7.2.2 Architecture of the Neural Network

Layer (type)	**Output Shape**	**Param #**
input_1 (InputLayer)	(None, 1, 1)	0
input_2 (InputLayer)	(None, 1, 1)	0
input_3 (InputLayer)	(None, 1, 1)	0
input_4 (InputLayer)	(None, 1, 1)	0
concatenate (Concatenate)	(None, 1, 4)	0
dense (Dense)	(None, 1, 6)	30
lstm (LSTM)	(None, 1, 12)	912
lstm_1 (LSTM)	(None, 1, 12)	1200
lstm_2 (LSTM)	(None, 1, 18)	2232
lstm_3 (LSTM)	(None, 1, 24)	4128
lstm_4 (LSTM)	(None, 1, 18)	3096
lstm_5 (LSTM)	(None, 1, 12)	1488
lstm_6 (LSTM)	(None, 1, 12)	1200
lstm_7 (LSTM)	(None, 6)	456
dense_1 (Dense)	(None, 1)	7

Table 1: Neural Network Architecture


Input Layer ∈ $\mathbb{R}^6$
Hidden Layer ∈ $\mathbb{R}^{12}$
Hidden Layer ∈ $\mathbb{R}^{12}$
Hidden Layer ∈ $\mathbb{R}^{18}$
Hidden Layer ∈ $\mathbb{R}^{24}$
Hidden Layer ∈ $\mathbb{R}^{18}$
Hidden Layer ∈ $\mathbb{R}^{12}$
Hidden Layer ∈ $\mathbb{R}^{11}$
Output Laye

NN WEIGHTS FIRST ENTRY

```
[-1.3158753  -0.4892516   0.7667988   0.0288299  -0.62861747 -0.34718516]

[ 0.12609749 -1.7035676   0.24501395  0.30555445 -0.07373061 -1.577009  ]

[-0.24600908 -0.03532389  0.00995408 -0.09426929  0.021626    0.85759133]

[ 1.1419278   0.80790037  0.9138882  -0.6260078  -0.83464617 -0.6504741 ]
```

7.2.2.1 Analysis of the Neural Network Weights and Structure

The structure of the neural network model can be described as follows:

- The model is designed with four distinct input layers, each corresponding to a specific financial feature: VPIN, intensities, order book ratios, and returns.
- These individual input layers are then concatenated to form a unified layer.
- The concatenated layer is then connected to a dense hidden layer which comprises 6 neurons. The activation function chosen for this hidden layer is the ReLU (Rectified Linear Unit).

Given this structure, the provided weight matrix for the hidden layer has dimensions 4×6, meaning each of the four input features connects to all six neurons in the dense layer.

A closer look at the weights reveals:

1. **VPIN**: The first row of the weight matrix corresponds to the VPIN feature. The weights here are both negative and positive, hinting at a complex relationship between VPIN and the patterns captured by the six neurons. Notably, a weight of -1.3158753 suggests a potential inverse relationship between this feature and the activation of the corresponding neuron.
2. **Intensities**: The weights for the second feature, intensities, are dominantly neg-

ative. A standout value of -1.7035676 indicates that higher intensity values could suppress the activation of its linked neuron, or vice versa.

3. **Order Book Ratios**: For the order book ratios, the third row, the weights are generally of smaller magnitude, indicating a more nuanced influence on the dense layer's neurons.

4. **Returns**: The last row, representing the returns feature, displays mixed weights. This implies that returns have a multifaceted influence on the patterns recognized by the neurons. Both positive and negative weights, such as 1.1419278 and -0.83464617 respectively, underline this complexity.

7.3 Comparative Analysis of Neural Network and OLS Metrics

Upon analyzing the metrics for both the Neural Network (NN) and the Ordinary Least Squares (OLS) model, several observations can be made:

1. **Accuracy**: The NN model exhibits an accuracy of 52.77%, which is notably higher than the 46.29% accuracy of the OLS model. This suggests that the NN model might be better suited for capturing the underlying complexities of the dataset and making more precise predictions.

2. **Mean Absolute Error (MAE)**: Both models have a relatively close MAE, with the NN model registering 0.8843 and the OLS model at 0.8290. Although the difference is marginal, the slightly lower MAE for the OLS model indicates that, on average, its predictions deviate less from the true values.

3. **Mean Squared Error (MSE)[20]**: The MSE values for both models are also in close proximity, with NN at 1.3017 and OLS at 1.2128. A smaller MSE value for the OLS model suggests it may be slightly better at avoiding large errors in its predictions.

4. **OLS Parameters**: The intercept and coefficients for the OLS model offer insights into the model's understanding of the relationships between variables. A positive coefficient suggests a direct relationship, while a negative one indicates an inverse relationship.

An important observation from real-world application is the models' performance during market fluctuations. It appears that both models, especially the Neural Network, exhibit a heightened ability to forecast market crashes effectively. This aptitude is commendable and can be invaluable for risk mitigation and strategic positioning during turbulent times. However, during periods of market stability or inactivity, these models seem to be less reliable as predictors. In quieter market phases, the inherent noise and randomness might overshadow any discernible patterns, making predictions more challenging. It is, therefore, advisable to exercise caution and possibly integrate additional predictive indicators when the market is relatively stable.

8 Conclusion

In this thesis, we have applied a different approach to the analysis of the Bitcoin market, incorporating both statistical methods and machine learning models. Our exploration of the order book's statistical properties, alongside the application of indicator like VPIN and the Hawkes process, has been proved to be important as input for our forecasting models.
The Neural Network performed better in terms of accuracy, reaching 52.77%. On the other hand, the OLS model was slightly more consistent, although it wasn't as accurate. This means it's more reliable for straightforward predictions.

We also noticed something interesting when we looked at how these models work in real-life market situations. Both models, especially the NN, were really good at predicting big market changes, like crashes. This is great for reducing risk and making smart moves during uncertain times.

References

[1] Baeldung. *Gradient Descent vs. Ascent: A Complete Guide*. URL: https://www.baeldung.com/cs/gradient-descent-vs-ascent.

[2] P. L. Bartlett, P. M. Long, G. Luosi, and A. Tsigler. "Benign Overfitting in Linear Regression". In: *Proceedings of the National Academy of Sciences PNAS* (2020). DOI: https://doi.org/10.1073/pnas.1907378117.

[3] S. Borman. "The Expectation Maximization Algorithm: A short tutorial". In: *Unpublished* (2004). URL: https://www.lri.fr/~sebag/COURS/EM_algorithm.pdf.

[4] J.-P. Bouchaud. "Statistical properties of stock order books: empirical results and models". In: *Quantitative Finance* 2 (2002), pp. 251–256.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood From Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society, Series B* 39 (1977), pp. 1–38.

[6] Machine Learning Glossary Docs. *Loss Functions*. URL: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

[7] Binance Documentation. Binance. URL: https://binance-docs.github.io/.

[8] D. Easley, R. F. Engle, M. O'Hara, and L. Wu. "Time-Varying Arrival Rates of Informed and Uninformed Traders". In: *Journal of Financial Econometrics* 6.2 (2008), pp. 171–207.

[9] D. Easley, M. Lopez de Prado, and M. O'Hara. "An Improved Version of the Volume-Synchronized Probability of Informed Trading: A Comment". In: *Critical Finance Review* 6 (2017), pp. 377–379.

[10] C. D. Ghilani and P. R. Wolf. *Adjustment Computations: Spatial Data Analysis.* (2006).

[11] A. G. Hawkes. "Spectra of Some Self-Exciting and Mutually Exciting Point Processes". In: *Biometrika* 58.1 (1971), pp. 83–90.

[12] F. Hayashi. *Econometrics*. Princeton University Press, (2000).

[13] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle. *GNSS – Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more*. Springer Wien New York, (2007).

[14] J. Kenney and E. S. Keeping. *Mathematics of Statistics*. Van Nostrand Reinhold Inc, (1963).

[15] W. Kenton. *What Is an Order Book? Definition, How It Works, and Key Parts.* Investopedia. Reviewed by Brock, T. (2022). URL: `https://www.investopedia.com/terms/o/order-book.asp`.

[16] E. Lewis and G. Mohler. "A Nonparametric EM Algorithm for Multiscale Hawkes Processes". In: *Journal of Nonparametric Statistics* (2011), pp. 1–6.

[17] K. Obral, J. Barry, and J. Kang. *Simulation, Estimation and Applications of Hawkes Processes.* (2016).

[18] Y. Ogata. “Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes”. In: *Journal of the American Statistical Association* (1981), pp. 9–27.

[19] *Ordinary Least Squares Regression (OLS).* XLSTAT. URL: https://www.xlstat.com/en/solutions/features/ordinary-least-squares-regression-ols.

[20] Probability-Course. *Mean Squared Error (MSE).* URL: www.probabilitycourse.com.

[21] P. Ranganathan, C. Pramesh, and R. Aggarwal. “Common Pitfalls in Statistical Analysis: Logistic Regression”. In: *PMC (PubMed Central)* (2017), pp. 148–151.

[22] S. Ross. *Introduction to Probability Models.* 11th. Academic Press, (2014).

[23] H. Saleh and J. A. Layous. “Machine Learning – Regression”. Available at: https://www.researchgate.net/publication/357992043. PhD thesis. Damascus, Syrian Arab Republic: Higher Institute for Applied Sciences, Technology, Department of Electronic, and Mechanical Systems, (2022). DOI: 10.13140/RG.2.2.35768.67842.

[24] A. F. Schmidt and C. Finan. “Linear Regression and the Normality Assumption”. In: *Journal of Clinical Epidemiology* 98 (2017), pp. 146–151.

[25] S. E. Shreve. *Stochastic Calculus for Finance II: Continuous-Time Models.* Springer, (2004).

[26] D. Stansbury. *Derivation: Derivatives for Common Neural Network Activation Functions.* (2020). URL: https://dustinstansbury.github.io/theclevermachine/derivation-common-neural-network-activation-functions.

[27] Study-Machine-Learning. *Mathematics Behind the Neural Network.* URL: https://studymachinelearning.com/mathematics-behind-the-neural-network/.

[28] A. Veen and F. P. Schoenberg. "Estimation of spacetime branching process models in seismology using an emtype algorithm". In: *Journal of the American Statistical Association* 103 (2008), pp. 614–624.

[29] T. Vipond. *What are Trade Orders?* Corporate Finance Institute. URL: https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/trade-orders/.

[30] Wikipedia. *Big O notation.* URL: https://en.wikipedia.org/wiki/Big_O_notation.

[31] Wikipedia. *Gradient descent.* URL: https://en.wikipedia.org/wiki/Gradient_descent.

[32] G. Xu. *GPS: Theory, Algorithms and Applications.* Springer, (2007).

[33] L. Xu and Michael I. Jordan. "On Convergence Properties of the EM Algorithm for Gaussian Mixtures". In: *Neural Computation* 8 (1996).

[34] D. Zwillinger. "Standard Mathematical Tables and Formulae". In: *Chapman & Hall/CRC* (1995).